Sequence-Aware Online Container Scheduling with Reinforcement Learning in Parked Vehicle Edge Computing

Jianqiu Wu, Jianxiong Guo, *Member, IEEE*, Zhiqing Tang, *Member, IEEE*, Chuanwen Luo, Tian Wang, *Senior Member, IEEE*, and Weijia Jia, *Fellow, IEEE*

***Abstract*—Intelligent vehicles, often parked for long periods, are ideally suited to serve as computational nodes to expand the Mobile Edge Computing (MEC) infrastructure, with containerization significantly enhancing the system's load balancing, self-healing, resource isolation, and security. However, fluctuations in task demand and frequent container image downloads during peak hours create high loads on containerized nodes, as multiple mobile devices offload tasks simultaneously, leading to significant processing delays. Many existing studies make the simplified assumption of predefined patterns of task arrivals, which overlooks this issue and makes suboptimal decisions. In this paper, we consider a Parked Vehicles (PVs)-extended MEC scenario, where multiple devices request services on PVs functioning as edge servers, all controlled by a central base station. Task arrivals follow observed patterns based on long-term trends, such as peak and off-peak periods, resembling realistic arrival patterns rather than predefined ones. To optimize task offloading by identifying these patterns, we propose the Sequence-Aware Task Scheduling (SATS) algorithm, which is a policy gradient-based deep reinforcement learning approach that integrates Transformer and LSTM architectures to capture patterns in time-series task arrivals and relationships between nodes in a collaborative and containerized environment, thereby enhancing the efficiency of online task scheduling. The primary objective of SATS is to optimize the task offloading policy and minimize delay and energy consumption for all devices and PVs. Extensive numerical comparisons against baselines demonstrate the effectiveness and advantages of our algorithm.**

***Index Terms*—Parked Vehicles, Vehicles Edge Computing, Mobile Edge Computing, Container, Reinforcement Learning, LSTM, Transformer**

I. Introduction

In recent decades, the significant increase in electric vehicles has made them a key component of the Internet of Things (IoT). With many vehicles parked for considerable periods, Parked Vehicles (PVs) are well-suited as edge computing nodes, providing computational resources to nearby devices and enhancing efficiency. To optimize resource utilization on PVs, containers are widely adopted [1]–[3]. These studies demonstrate the ability of containers to enhance the effective use of PVs. Leveraging containers avoids the high financial and time costs of deploying dedicated Vehicles Edge Computing (VEC) servers, while still supporting a range of computation-intensive applications. As shown in Figure 1, containers operate at the Operating System (OS) level, enhancing resource efficiency by creating isolated user-space instances. Additionally, containers provide faster start-up speeds and lower hardware and operational costs compared to virtual machines. Therefore, containerization offers an effective and scalable solution for optimizing PVs in VEC systems.

Before running a container, a local image file containing the code, binaries, system tools, and configuration files must be available, and the download time of multiple images could be time-consuming, especially for delay-sensitive tasks that are often short-lived. Several studies have proposed task-offloading algorithms for containerized scenarios. Liu *et al.* [4] introduced a Deep Reinforcement Learning (DRL)-based approach to address computation offloading and resource allocation strategies within VEC networks. Similarly, Lu *et al.* [5] tackled the task offloading challenge in a container-based edge computing framework, leveraging the Proximal Policy Optimization (PPO) algorithm. Notably, [4] assumes that each device generates exactly one task in every time

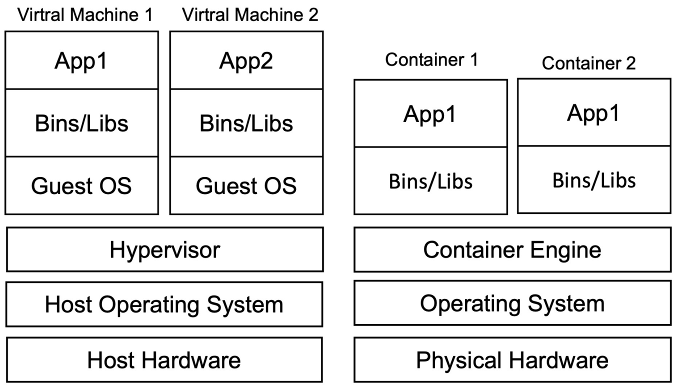


Fig. 1. Comparison of hypervisor-based and container-based virtualizations.

Jianqiu Wu is with the Guangdong Key Lab of AI and Multi-Modal Data Processing, Department of Computer Science, BNU-HKBU United International College, Zhuhai 519087, China. (E-mail: jqwuhelen@qq.com)

Jianxiong Guo and Weijia Jia are with the Advanced Institute of Natural Sciences, Beijing Normal University, Zhuhai 519087, China, and also with the Guangdong Key Lab of AI and Multi-Modal Data Processing, BNU-HKBU United International College, Zhuhai 519087, China. (E-mail: jianxiongguo@bnu.edu.cn; jiawj@bnu.edu.cn)

Zhiqing Tang and Tian Wang are the Advanced Institute of Natural Sciences, Beijing Normal University, Zhuhai 519087, China. (E-mail: zhiqingtang@bnu.edu.cn; cs_tianwang@163.com)

Chuanwen Luo is with the School of Information Science and Technology, Beijing Forestry University, Beijing 100083, China, and also with the Engineering Research Center for Forestry-Oriented Intelligent Information Processing of National Forestry and Grassland Administration, Beijing 100083, China. (E-mail: chuanwenluo@bjfu.edu.cn)

(Corresponding author: Jianxiong Guo.)

Manuscript received April xxxx; revised August xxxx.

slot, which simplifies task arrival patterns by treating them as deterministic. While [5] models tasks as arriving in a random sequence, allowing for more variability in task arrival times, but potentially ignoring the impact of structured patterns such as peak hours or bursts of demand.

In practice, however, PVs often face limited processing capacities. When a large number of mobile devices simultaneously offload their tasks to the same PV, the PV's load may become excessive, further extending image download times and leading to increased processing delays. Furthermore, downloading multiple necessary images while simultaneously uninstalling existing ones is highly time-consuming due to storage limitations in PVs. This adds additional burden and further prolongs the overall task completion time. An effective solution must consider the image repositories of PVs in the current time slot and adapt image requests during peak hours, while also accounting for time-series patterns to anticipate future demand and minimize unnecessary costs. While existing approaches, such as those in [6]–[9] rely on traditional methods, struggling to capture the dynamic, time-varying nature of task allocation and resource distribution. The work in [10]–[12] employs Reinforcement Learning (RL) techniques that primarily focus on optimizing task offloading policies to balance computational loads or enhance resource allocation efficiency. However, these approaches overlook the critical impact of dynamic storage strategies and temporal variations, which significantly influence overall system performance.

Consequently, we develop the Sequence-Aware Task Scheduling (SATS) algorithm, a policy gradient-based RL method that leverages a Transformer to capture the dynamic characteristics of the PV environment, enriching our policy network with real-time resource information. Additionally, by incorporating a Long Short-Term Memory (LSTM) model, historical task trends can be analyzed to enhance predictive accuracy, thereby enabling the model to make informed decisions for the upcoming peak hours. The memory cells and gating mechanisms of LSTMs help address issues like vanishing gradients, which are crucial for efficient time-series prediction [13]. Existing studies propose caching strategies using LSTM [14], [15] and Transformer architectures [16], respectively, to predict content demand, maximize cache efficiency, and predict energy arrival process in dynamic MEC environments. These studies demonstrate the effectiveness of leveraging LSTM and Transformer models to tackle time-series challenges and improve resource utilization in MEC systems. In MEC networks, LSTMs enhance performance by predicting network traffic, optimizing resource allocation, and reducing latency. Transformers, with their self-attention mechanism and parallel processing capabilities, capture global relationships across multiple PVs. Together, these technologies form a comprehensive framework that improves task offloading policies and enhances SATS efficiency.

Our containerized PV-based VEC system comprises a diverse set of request devices and PVs, which download images from the Base Station (BS), serving as the central controller. The BS executes the SATS algorithm and facilitates wireless communication between PVs and devices. The model's ability to account for task arrivals following discernible patterns further enhances its practical applicability and overall performance. Experimental results demonstrate SATS's effectiveness in generating efficient task offloading strategies, reducing image download time and energy costs, and achieving superior optimization outcomes compared to baseline methods. The key contributions of our research are summarized as follows:

1) We model task scheduling in VEC systems by framing it as a patterned task allocation problem. Containerization is employed to improve computation and isolation in the VEC environment, equipping PVs with containers to enable lightweight, scalable offloading services. PVs download appropriate images from the base station to execute tasks offloaded from request devices. Unlike existing methods that assume static or deterministic task arrivals, our approach enhances the accuracy and applicability of task offloading within PVs.
2) To address the issues of neglected task patterns and the failure of existing models to recognize and adapt to the complex and evolving relationships among PVs, we integrate LSTM and Transformer models into our system. This integration captures the historical patterns of task arrivals and deepens our understanding of the image repository among PVs. By doing so, we aim to significantly enhance system performance, particularly by reducing latency costs and improving the reliability of task offloading policies.
3) We propose a policy gradient-based RL algorithm named SATS, specifically designed for VEC networks, focusing on addressing the latency and limited computational power of PVs. This leads us to establish an optimization problem, which seeks to reduce both energy and time costs within the network.
4) We evaluate our proposed algorithm against three baseline methods. The results demonstrate that our approach achieves better performance and significantly improves efficiency by recognizing time patterns with peak hours. This method can lead to up to a 44% reduction in time costs and a 37% reduction in energy costs, proving to be highly effective.

The remainder of this paper is organized as follows: Section II reviews the relevant literature. Section III outlines the architecture of the VEC system. Section IV introduces the SATS algorithm and formulates the problem of minimizing time and energy costs. Section V presents the simulation results. Finally, the paper concludes with Section VI.

II. Related work

Vehicles Edge Computing. Ren *et al.* [11] propose a blockchain-based trust management framework for VEC networks, using DRL to optimize vehicular service offloading and migration, enhancing trust and resource allocation efficiency. Fan *et al.* [6] present a game-theoretic approach for task offloading and resource allocation in VEC, incorporating edge-to-edge cooperation to optimize load balancing and reduce latency. Liu *et al.* [18] proposed a RL-based solution to maximize network utility, but neglected the network's temporal dynamics. Collectively, these studies highlighted the inherent

TABLE I
DIFFERENCE BETWEEN OUR SCHEME AND OTHER RELATED SCHEMES.

Scheme	Vehicle Assistance	Container Involved	Reinforcement Learning	Transformer Usage	Task Arrival Patterns
[17]	N/A	×	×	×	×
[10]	Parked Vehicles Assistance	×	✓	×	×
[11]	N/A	×	✓	×	×
[6]	N/A	×	×	×	×
[7]	Moving Vehicles Assistance	×	×	×	×
[8]	Parked Vehicles Assistance	✓	×	×	×
[9]	Parked Vehicles Assistance	✓	×	×	×
[1]	Parked Vehicles Assistance	✓	×	×	×
[12]	Parked Vehicles Assistance	✓	✓	×	×
Our Scheme	Parked Vehicles Assistance	✓	✓	✓	✓

challenges associated with mobile vehicles within VEC systems. Consequently, they indicated that PVs might provide a more stable and efficient alternative for VEC applications, potentially enhancing the reliability and effectiveness of edge computing environments. Recent studies have leveraged conventional vehicles as edge computing nodes in various applications. Feng *et al.* [19] utilized mobile vehicles' idle computational capacity for offloading services, eliminating the need for fixed infrastructure. This approach contrasted with using PVs, where Hou *et al.* [20] highlighted in their introduction of vehicular fog computing, exploiting PVs as stationary nodes for data relaying and processing. Expanding on this concept, Fan *et al.* [21] integrated both mobile and parked vehicles with base stations to enhance the MEC framework, focusing on reducing priority-weighted task processing delays. They proposed an iterative solution using generalized Benders decomposition, complemented by a heuristic algorithm for precision in near-optimal solutions. Ge *et al.* [22] proposed a two-stage service migration algorithm for Parked Vehicle Edge Computing (PVEC) networks. This algorithm breaks down the original problem into two distinct stages: service migration between Service Providers (SP) and the selection of serving parked vehicles in parking lots.

Involvement of Containers. Due to its lightweight nature and ease of deployment, containerization has become a promising approach in MEC environments. Various studies have been conducted to leverage this technology, including efforts to achieve energy-efficient task selection and scheduling within edge networks [23], to facilitate rapid container deployment and simplify management for low-latency video analytics close to users [24], and to assess container performance in edge computing environments through real experiments [25], [26]. Tang *et al.* [27] developed a container scheduling algorithm that comprehends the complex dependencies between layers and images, aiming to reduce the overall task completion time. Nguyen *et al.* [2] advanced the exploration of container technology on PV servers, and developed a utility-aware heuristic to reduce system costs while ensuring service quality. However, they assumed PVs autonomously generate containers, an impractical expectation as [1]. They addressed a task-container matching problem using a matching game to facilitate on-demand offloading, yet underestimated PVs' capability to update images for diverse tasks.

To the best of our knowledge, existing research has not effectively modeled the integration of PVs and container technologies. Our model design and the development of the PV-container system address the ongoing challenges of container management of PVs, highlighting the urgent need for continued innovation to fully leverage these technologies' potential. As shown in Table I, we delineate the distinctions between our approach and other methodologies. Notably, our strategy provides a more comprehensive model, effectively incorporating the advantages of container technology, task arrival patterns, and PV interconnection to enhance the overall efficiency and responsiveness of VEC systems.

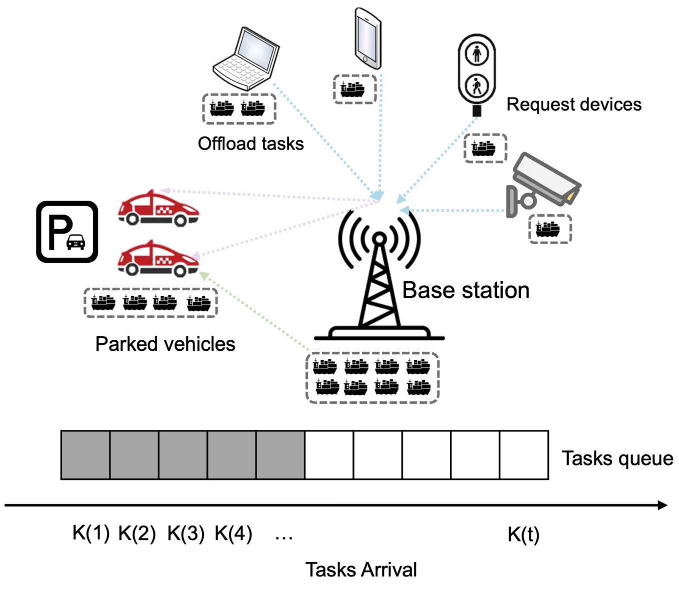


Fig. 2. An edge computing framework where parked vehicles serve as nodes to which devices offload tasks, making use of the PVs' unused computational resources. PVs and devices can download the necessary container images from the base station. Tasks arrive in each time slot, and $K(1)$ denotes the set of tasks arriving in time slot 1.

III. System Model & Problem Formulation

In this section, we first define our system model, transmission and computing latency, and then present the problem formulation step by step.

A. System Model

As shown in Figure 2, we introduce a container-based VEC framework, tailored for parking lot scenarios, enabling PVs to download essential images to facilitate task offloading in VEC environments. In this scenario, a BS, denoted by N_c, serves as the central node, providing service coverage to N PVs and to devices numbered sequentially from $N+1$ to M. The system time is divided into discrete time frames of equal lengths, denoted by $t \in \mathcal{T} = \{1, 2, \cdots\}$. The collection of PVs is denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$. Each PV n is equipped with a computing unit defined by bandwidth B_n, maximum storage capacity D_n, CPU frequency F_n, maximum memory M_n, and remaining memory $\mu_n(t)$ at time slot t. The coordinate of the PV n is given as (x_n, y_n), collectively defining the vehicle n as tuple $n(t) = (B_n, D_n, F_n, M_n, \mu_n(t), x_n, y_n)$. Additionally, a variety of devices, represented as the set $\mathcal{M} = \{N+1, N+2, \ldots, M\}$, connect to this network and offload tasks requiring container execution to these vehicular computing units. These devices are similarly specified by attributes including CPU frequency F_m, bandwidth B_m, maximum storage capacity D_m, maximum memory size M_m, and remaining memory $\mu_m(t)$ at time slot t. Their coordinates at time slot t are given by $(x_m(t), y_m(t))$. Collectively, these specifications form a tuple represented as $m(t) = (B_m, D_m, F_m, M_m, \mu_m(t), x_m(t), y_m(t))$. For task processing, a PV must initiate a container, which relies on locally accessible images. To facilitate uninterrupted task execution, the necessary images must be pre-fetched and locally stored. The repository of images is symbolized by $\mathcal{I} = \{1, 2, \ldots, I\}$, with each image i corresponding to a distinct container, whose size is s_i. It is presumed that requesting a container is synonymous with requesting the associated image, all of which are maintained at the BS.

We define $x_n^i(t) \in \{0, 1\}$ to indicate the presence of image i on a PV n at the beginning of time slot t, where $x_n^i(t) = 1$ means that image i is stored on PV n, and $x_n^i(t) = 0$ signifies its absence. A similar notation, $x_m^i(t) \in \{0, 1\}$, is used for devices, where $x_m^i(t) = 1$ indicates that image i is stored on device m, and $x_m^i(t) = 0$ indicates it is not. It is important to note that both vehicles and devices possess limited storage capacities for these images, ensuring that the stored images do not exceed the available disk space on any vehicle or device. We formally articulate this constraint as follows:

$$C1: \sum_{i \in \mathcal{I}} x_n^i(t) \times s_i \leq D_n, n \in \mathcal{N}, t \in \mathcal{T} \tag{1}$$

$$C2: \sum_{i \in \mathcal{I}} x_m^i(t) \times s_i \leq D_m, m \in \mathcal{M}, t \in \mathcal{T}. \tag{2}$$

In each time slot t, each device can send multiple computational tasks, and the collection of tasks emanating from devices during a specific time slot t is represented as $K(t) = \{1, 2, \cdots, K_t\}$. The task $k = (d_k, m_k, f_k, i_k, t_k^{arrive}, t_k^{ddl}, q_k, b_k)$, where d_k is the size of the task k, m_k is the memory needed by the task k, f_k is the total number of CPU cycles required to accomplish the computation task, i_k is the specific image required for a task, selected from a limited assortment of available images. Meanwhile, q_k represents the device that generates the task k, b_k is the allocated bandwidth of the task, and t_k^{ddl} is the maximum tolerance deadline for completing this task. Here, We employ the indicator $\alpha_k^n(t) = 1$ to signify that task k is designated for offloading to PV n. Conversely, when task k is assigned for local execution, implying it is allocated to the originating device q_k, we use another indicator $\beta_k(t) = 1$. This setup is governed by the following constraints:

$$C3: \alpha_k^n(t) \in \{0, 1\}, k \in K(t), n \in \mathcal{N}, t \in \mathcal{T}. \tag{3}$$

We assume that tasks from each device cannot be subdivided into sub-tasks. Therefore, for any task $k \in K(t)$, the offloading must be directed to a single destination, either one of the PVs or executed locally on the device itself. This is expressed by the constraint:

$$C4: \sum_{n \in \mathcal{N}} \alpha_k^n(t) + \beta_k(t) = 1, k \in K(t), t \in \mathcal{T}. \tag{4}$$

This constraint ensures that for any given task k, either $\alpha_k^n = 1$ for PV n, indicating offloading to that node, or $\beta_k = 1$ signifying local execution.

B. Offloading to PVs

Task Transmission Latency. When a device queues a task request at the BS, it transmits only the task information rather than the entire task payload. Given that these attribute files are significantly smaller than the tasks themselves, the associated transmission cost is minimal and can generally be disregarded. Consistent with several previous studies [28]–[31], this work also disregards the latency associated with transmitting results and the offloading policy back to the devices. This decision is based on the observation that the size of the resultant data is generally much smaller than that of the initial tasks. Furthermore, the communication duration between devices and PVs is not considered in our analysis. The BS sequentially processes the queued requests and executes a strategy to assign the appropriate PV for each task allocation. According to Shannon's theory, the data transfer rate from device m to PV n at time slot t can be calculated as follows:

$$r_{m,n}(t) = B_n \log\left(1 + \frac{p h_{m,n}^2(t)}{\sigma^2}\right), \tag{5}$$

where B_n is allocated to represent the bandwidth of PV n and p serves to denote the transmission power. The interaction in terms of channel gain between device m and PV n at time slot t, expressed as $h_{m,n}(t)$, is calculated utilizing the Free Space Path Loss (FSPL) equation, as outlined in [32]. The equation is given by $h_{m,n}(t) = 20\log_{10}(d_{m,n}(t)) + 20\log_{10}(f) + 20\log_{10}\left(\frac{c}{4\pi}\right)$, where $d_{m,n}(t)$ delineates the Euclidean distance between m and n at time slot t, computed by $\sqrt{(x_m(t) - x_n)^2 + (y_m(t) - y_n)^2}$, and f represents the frequency of the channel. Furthermore, σ^2 is employed to denote the Gaussian noise power. Factoring in these variables, the transmission latency for task k at the time slot t can be explicated as follows:

$$T_k^{p,tr}(t) = \sum_{n \in \mathcal{N}} \frac{d_k}{r_{q_k,n}(t)} \cdot \alpha_k^n(t). \tag{6}$$

Image Download Latency. Each PV n is allocated a specific storage capacity, denoted by D_n, and the memory capacity is denoted by M_n. This capacity and memory constraint ensures that the total size of all containers hosted on a single PV does not exceed D_n and the memory utilized by these containers does not surpass M_n. To effectively regulate this constraint, each PV employs a Least Frequently Used (LFU) strategy for its container storage queue. In instances where an incoming task requires a container absent from the current queue and existing storage or memory proves inadequate, the protocol mandates the removal of the least accessed container within the queue. This measure is undertaken to reclaim necessary space. This iterative process of eviction and reallocation persists until the PV can feasibly accommodate the requisites of the new container of arriving tasks. It is assumed that only one image can be downloaded at one time, and thus each node has an image download queue. If a new image needs to be downloaded, it should be checked whether the remaining storage and memory are enough for the image. If not, existing images have to be removed until there is enough space. Then the image can be added to this queue and try to download the next image if required.

The term $T_n^{queue}(t_k^{arrive})$ represents the queuing latency at PV n when the task k arrives. Consequently, the cumulative image download latency incurred by processing task k is shown as follows: $T_k^{p,down}(t) =$

$$\sum_{n\in\mathcal{N}} \left(\frac{j_{i_k}}{r_{N_c,n}(t)} + T_n^{queue}(t_k^{arrive})\right)\cdot \alpha_k^n(t)\cdot(1-x_n^{i_k}(t)). \quad (7)$$

Computation latency. The computation time is the processing time of the task k on a designated PV, which can be obtained as follows:

$$T_k^{p,com}(t) = \sum_{n\in\mathcal{N}} \frac{f_k}{F_n}\cdot \alpha_k^n(t), \quad (8)$$

where f_k is the CPU cycles requested by the task k and F_n is the computing power of the PV n. We assume that the number of threads available in the PV exceeds the number of tasks, ensuring that the CPU frequency remains unaffected.

Energy cost. Given the task transmission latency and image download latency, with p_n^{tran} and p_n^{com} representing the transmission power and computation power of PV n, we can define the computation and the transmission energy cost for task k as follows:

$$E_k^{p,com}(t) = T_k^{p,com}(t)\sum_{n\in\mathcal{N}} p_n^{com}\cdot \alpha_k^n(t), \quad (9)$$
$$E_k^{p,tr}(t) = (T_k^{p,tr}(t) + T_k^{p,down}(t))\sum_{n\in\mathcal{N}} p_n^{tr}\cdot \alpha_k^n. \quad (10)$$

C. Local Computing

In our architecture, tasks created locally on purpose-built devices like cameras or traffic lights still necessitate container downloads for execution. Given that these devices are tailored for specific functions such as monitoring and traffic control, they do not inherently possess the flexibility to manage diverse tasks. Consequently, to address this limitation, containers are employed to establish a uniform and isolated execution environment, ensuring seamless operation across these specialized devices, which will be more efficient.

Image Download Latency. The image download process in the local computing scenario is subject to similar constraints as those in the offloading to PVs scenario and we will not repeat the explanation. If task k is assigned to be processed locally, the download latency can be: $T_k^{l,down}(t) =$

$$\left(\frac{j_k}{r_{N_c,q_k}(t)} + T_{q_k}^{queue}(t_k^{arrive})\right)\cdot \beta_k(t)\cdot(1-x_{q_k}^{i_k}(t)). \quad (11)$$

Local Computation latency. The local computation latency can be calculated as follows:

$$T_k^{l,com}(t) = \frac{f_k}{F_{q_k}}\cdot \beta_k(t), \quad (12)$$

where F_{q_k} represents the computing power of the device that generates task k.

Energy cost. Considering the computation latency and image download latency, the energy required for computing and transmitting task k can be defined as:

$$E_k^{l,com}(t) = T_k^{l,com}(t)p_{q_k}^{com}\cdot \beta_k(t), \quad (13)$$
$$E_k^{l,tr}(t) = T_k^{l,down}(t)p_{q_k}^{tr}\cdot \beta_k(t). \quad (14)$$

D. Problem Formulation

To simplify the formulation, we first identify the offloading policy at time slot t, which is defined as the set $\boldsymbol{\alpha}(t) = \{(\alpha_k^n(t), \beta_k(t)) : k \in K(t), \forall n \in \mathcal{N}\}$. Given the offloading policy set $\boldsymbol{\alpha}(t)$ at time slot t, the total latency for executing task k is represented as follows:

$$\begin{aligned} T_k^{total}(t) = T_k^{p,tr}(t) + (T_k^{p,down}(t) + T_k^{l,down}(t)) \\ + (T_k^{p,com}(t) + T_k^{l,com}(t)). \end{aligned} \quad (15)$$

Similarly, the total energy cost of task k execution can be denoted as: $E_k^{total}(t) =$

$$(E_k^{p,tr}(t) + E_k^{p,com}(t)) + (E_k^{l,tr}(t) + E_k^{l,com}(t)). \quad (16)$$

We aim to minimize the overall task completion time and energy cost from a long-term perspective. We aim to identify the optimal strategy that minimizes the overall cost while adhering to constraints. We define the weight parameter w_t to adjust the balance in optimization between time cost and energy cost. Therefore, the problem is defined as follows:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}(t)} \sum_{t\in\mathcal{T}}\sum_{k\in K(t)} \left(w_t\cdot T_k^{total}(t) + (1-w_t)\cdot E_k^{total}(t)\right) \\ &s.t.\ C1 - C4; \\ &C5: \mu_n(t) + \sum_{k\in K(t)} \alpha_k^n(t)\cdot m_k \le M_n, n\in\mathcal{N}, t\in\mathcal{T}; \\ &C6: \mu_m(t) + \sum_{k\in K(t)} \beta_k(t)\cdot \mathbb{I}[q_k = m]\cdot m_k \le M_m, \\ &\quad m\in\mathcal{M}, t\in\mathcal{T}; \\ &C7: T_k^{total}(t) + t_k^{arrive} \le t_k^{ddl}, k\in\mathcal{K}(t), t\in\mathcal{T}. \end{aligned} \quad (17)$$

The constraints $C1$ to $C4$ have been previously explained. Constraint $C5$ specifies that each PV has a defined maximum memory capacity allocated for tasks and that the container does not utilize any memory before starting a task, allowing the memory cost of the container to be disregarded. Consequently, the memory used by tasks offloaded to the PV must not exceed this maximum capacity. Similarly, constraint $C6$ imposes identical limits on devices, ensuring that the memory capacity is not exceeded by the tasks they handle. Therefore, the total memory used by ongoing processing containers on any PV n or device m must not exceed this limit. As for constraint $C7$, each task must be completed by its specified deadline. The problem is formulated as a Mixed-Integer Nonlinear Programming (MINLP) problem, which is NP-hard. In this scenario, by carefully selecting the duration of each time slot, the probability of transitioning from one state to another based on resource demand remains nearly constant for a significant period and does not follow a uniform distribution [33]. Furthermore, the task arrival process and environmental updates exhibit a memoryless property, meaning that past events do not influence future outcomes. Given these characteristics, the scenario can be effectively represented using a Markov Decision Process (MDP) framework and addressed with RL techniques.

IV. Reinforcement Learning Solution

In this section, we start by defining the state space, action space, and reward function for the proposed problem. We then propose a policy gradient-based RL algorithm to address the problem. Additionally, to better capture the relationships between nodes, we introduce a transformer unit to embed the node features. Moreover, we utilize an LSTM component to incorporate the task history and patterns as input, predicting the number of tasks for the next step. We will first introduce the algorithm settings and then detail how the algorithm operates to solve our problem.

A. State, Action, and Reward Definition

The main components of RL are the agent and the VEC environment. The agent makes scheduling decisions. To train an agent, the state, action, reward, and policy are needed.

State Space. A state fully encapsulates the VEC environment, characterized by three key elements: computing nodes (encompassing both PVs and devices), user-requested tasks, and the history of task arrivals. To deeply analyze each node's dependency and consider computational resources, we partition the state of PVs and device nodes into two aspects: resource information and image information.

Node Information. The Node information encapsulates the resources and the locations of both the device nodes and the PVs, serving as edge nodes. The resources of each PV are characterized by its memory and storage capacities at time slot t, as well as by its CPU frequency, bandwidth, transition power, and computation power. These characteristics are collectively defined as follows:

$$\begin{aligned}\boldsymbol{s}_t^{node} = \{&\mu_1(t), \mu_2(t), \cdots, \mu_M(t),\\ &d_1(t), d_2(t), \cdots, d_M(t),\\ &x_1(t), x_2(t), \cdots, x_M(t),\\ &y_1(t), y_2(t), \cdots, y_M(t),\\ &P_1^{tr}, P_2^{tr}, \cdots, P_M^{tr}, P_1^{com}, P_2^{com}, \cdots, P_M^{com},\\ &B_1, B_2, \cdots, B_M, F_1, F_2, \cdots, F_M\}. \end{aligned} \quad (18)$$

where $d_n(t) = D_n - \sum_{i\in\mathcal{I}} x_n^i(t) \times s_i$ and $d_m = D_m - \sum_{i\in\mathcal{I}} x_m^i(t) \times s_i$.

Image Information. The image information, which includes both the download duration of images and image storage on each PV or device at a specific time slot, is crucial for the algorithm and significantly impacts the allocation of tasks to specific PVs. The download time for images required by task k is denoted by $T_k^{p,down}(t)$ for PVs and $T_k^{l,down}(t)$ for local devices. This parameter is essential for managing image data effectively and plays a key role in enhancing task scheduling performance. Additionally, the distribution of images across nodes holds significant importance. We represent the state of image download time with the notation s_t^{down}, which is defined as follows:

$$\begin{aligned}\boldsymbol{s}_t^{down} = \{&T_1^{p,down}(t), T_2^{p,down}(t), \cdots, T_{K_t}^{p,down}(t),\\ &T_1^{l,down}(t), T_2^{l,down}(t), \cdots, T_{K_t}^{l,down}(t)\}. \end{aligned} \quad (19)$$

For image storage information, given the limited types of images, we track whether each node possesses these images. This is represented by $x_n^i(t)$, as previously mentioned. Consequently, we obtain:

$$\boldsymbol{s}_t^i = \begin{bmatrix} x_1^1(t) & x_1^2(t) & \cdots & x_1^I(t)\\ x_2^1(t) & x_2^2(t) & \cdots & x_2^I(t)\\ \vdots & \vdots & \ddots & \vdots\\ x_N^1(t) & x_N^2(t) & \cdots & x_N^I(t)\\ \vdots & \vdots & \ddots & \vdots\\ x_M^1(t) & x_M^2(t) & \cdots & x_M^I(t) \end{bmatrix}. \quad (20)$$

Thus, the structure of the image information is outlined as follows:

$$\boldsymbol{s}_t^{image} = \boldsymbol{s}_t^{down} \oplus \boldsymbol{s}_t^i. \quad (21)$$

Task Information. The task state encompasses the necessary images for execution, as well as the resources requested by the task and constrains. Consequently, the task state is represented as follows:

$$\boldsymbol{s}_t^{task} = \cup_{k\in K(t)} \{d_k, m_k, f_k, i_k, t_k^{arrive}, t_k^{ddl}\}. \quad (22)$$

History Information. The historical information of time slot t is represented by $\boldsymbol{s}_t^{his}$, which records the number of task arrivals from several preceding time slots. We analyze these data using the lag time method, which entails looking back over several time slots. We then use an LSTM model to make predictions based on this historical context. In our simulation, the number of look-back steps is set to 5, following the methodology described in [34]. Combine all of the information, we can get the state:

$$\boldsymbol{s}_t = \boldsymbol{s}_t^{node} \oplus \boldsymbol{s}_t^{image} \oplus \boldsymbol{s}_t^{task} \oplus \boldsymbol{s}_t^{his}. \quad (23)$$

Action Space. The policy allocates tasks to PVs or the local device, taking into account the container configurations within them. It determines whether tasks should be processed locally or offloaded to a PV. Accordingly, the action space includes all PVs and devices, represented as the union of the set of

PVs $\mathcal{N}$ and the requesting device represented as follows:

$$\boldsymbol{a}_t \in (\mathcal{N} \cup \mathcal{M})^{K_t}. \tag{24}$$

Reward Function. Defining an appropriate reward function is critical for the success of RL algorithms. Given that offloading policies seek to balance energy consumption and execution time, relying solely on total cost may result in an unstable training process. Therefore, both the expected and actual cost of a task are incorporated into the reward function, which can be expressed as follows:

$$r_t = w_t \cdot \left(T_{ideal}^{total}(t) - \sum\nolimits_{k\in K(t)} T_k^{total}(t) \right) + (1-w_t) \cdot \left(E_{ideal}^{total}(t) - \sum\nolimits_{k\in K(t)} E_k^{total}(t) \right), \tag{25}$$

where $T_{ideal}^{total}(t) = \sum_{k\in K(t)} f_k / F_{max}$ defines the ideal latency for task execution in time slot t and F_{max} represents the maximum CPU frequency across all PVs and devices. Task completion times close to $T_{ideal}^{total}(t)$ yield higher rewards, reflecting efficiency in time utilization. Similarly, the ideal energy cost, $E_{ideal}^{total}(t) = T_{ideal}^{total}(t) \times p_{min}^{com}$, is calculated using p_{min}^{com}, the lowest power for computation across the PVs and devices. The reward decreases when the actual energy consumption exceeds $E_{ideal}^{total}(t)$, emphasizing the importance of minimizing energy use to enhance overall efficiency.

B. The Sequence-Aware Task Scheduling (SATS) Algorithm

The policy gradient-based SATS algorithm is introduced in this subsection, including policy optimization, transformed branch, and LSTM branch.

Overview. The architecture of the SATS algorithm is illustrated in Figure 3. The algorithm begins by observing the states of PVs, devices, and tasks from the environment. These features are processed through three distinct branches. The Transformer branch, designed to capture complex dependencies, takes $\boldsymbol{s}_t^{node} \oplus \boldsymbol{s}_t^{image}$ as input and produces $\boldsymbol{s}_t^{trans}$, as detailed in the Algorithm 1. A fully connected branch focuses on immediate state representation, receiving $\boldsymbol{s}_t^{task}$ as input and generating $\boldsymbol{s}_t^{fc}$. Meanwhile, the LSTM branch, which is used for recognizing temporal patterns, processes $\boldsymbol{s}_t^{his}$ and outputs $\boldsymbol{s}_t^{lstm}$. The outputs from these branches are concatenated with a copy of $\boldsymbol{s}_t^{node} \oplus \boldsymbol{s}_t^{image}$ to form the merged state $\boldsymbol{s}_t'$, which is then input into the policy network to determine the optimal scheduling actions. Actions taken by the policy network yield rewards, which serve as feedback for subsequent decision-making. The core of the SATS algorithm lies in policy optimization, which is facilitated through a policy gradient method. This method optimizes the policy directly by maximizing the expected return. The policy gradient method is utilized for its equilibrium between effectiveness and computational efficiency, improving learning stability and performance through measured policy adjustments, thus making it an apt selection for the SATS algorithm.

Policy Optimization. Policy gradient methods are designed for explicitly optimizing policies by evaluating the gradient of the expected rewards concerning policy parameters. Specifically, Actor-Critic algorithms, a subset of policy gradient techniques, merge the principles of policy enhancement (executed by the "Actor") with those of value function approximation (executed by the "Critic") [35]. In this structure, the Critic's role involves approximating the value function and guiding the Actor in making policy updates.

Algorithm 1: The SATS Algorithm.

```
1  Initialize policy parameters $\theta$ and value parameters $\phi$;
2  Initialize the sampling policy $\pi_{\theta_{old}}$ with $\theta_{old} \leftarrow \theta$;
3  for $Episode = 1, 2, \cdots$ do
4      Initialize replay memory $\mathcal{D} \leftarrow \emptyset$;
5      Reset environment;
6      // Sampling process;
7      foreach time slot $t \in \mathcal{T}$ do
8          Get current state $\boldsymbol{s}_t$;
9          $\boldsymbol{s}_t^{trans} \leftarrow$ Transformer($\boldsymbol{s}_t^{node} \oplus \boldsymbol{s}_t^{image}$);
10         $\boldsymbol{s}_t^{fc} \leftarrow$ FC($\boldsymbol{s}_t^{task}$);
11         $\boldsymbol{s}_t^{lstm} \leftarrow$ LSTM($\boldsymbol{s}_t^{his}$);
12         Get the
             $\boldsymbol{s}_t' = \boldsymbol{s}_t^{trans} \oplus \boldsymbol{s}_t^{fc} \oplus \boldsymbol{s}_t^{lstm} \oplus (\boldsymbol{s}_t^{node} \oplus \boldsymbol{s}_t^{image})$;
13         Select action $\boldsymbol{a}_t$ according to $\pi_{\theta_{old}}(\boldsymbol{a}_t|\boldsymbol{s}_t')$;
14         Execute action $\boldsymbol{a}_t$ and get the reward $r_t$;
15         Get the next state $\boldsymbol{s}_{t+1}$;
16         $\mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{s}_t', \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}')$;
17     // Using trajectory $\mathcal{D}$ to train networks;
18     for $epoch = 1, 2, \cdots$ do
19         Sample a batch $\mathcal{B}$ from $\mathcal{D}$ to update parameters;
20         Calculate $\mathcal{L}_{PPO}(\theta)$ by Eqn. (33);
21         Update actor parameter $\theta$ using Adam;
             Calculate $\mathcal{L}_{CRITIC}(\phi)$ by Eqn. (32);
22         Update critic parameter $\phi$ using Adam;
23     Update $\theta_{old} \leftarrow \theta$;
24 return Actor network $\pi_\theta$ and Critic network $V_\phi$;
```

Within the workings of policy gradient methods, accurately estimating the policy gradient is crucial, as it feeds into a stochastic gradient ascent algorithm. Central to this process is the advantage function, denoted by $A^\pi(\boldsymbol{s}_t', \boldsymbol{a}_t)$, which quantifies the relative merit of each action by assessing how it compares to the average action for a particular state. This function is instrumental in distinguishing the potential efficacy of different actions and is computed as follows:

$$A^\pi(\boldsymbol{s}_t', \boldsymbol{a}_t) = Q^\pi(\boldsymbol{s}_t', \boldsymbol{a}_t) - V^\pi(\boldsymbol{s}_t'), \tag{26}$$

where the value function $V^\pi(\boldsymbol{s}_t')$ represents the expected return for an agent starting in merging state $\boldsymbol{s}_t'$ and following policy π. This function is formulated as:

$$V^\pi(\boldsymbol{s}_t') = \mathbb{E}_\pi[G_t | S_t = \boldsymbol{s}_t'], \tag{27}$$

where $G_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$ and $\gamma \in (0, 1]$ is a discount factor. The state-action value function $Q^\pi(\boldsymbol{s}_t', \boldsymbol{a}_t)$ is defined as:

$$Q^\pi(\boldsymbol{s}_t', \boldsymbol{a}_t) = \mathbb{E}_\pi[G_t | S_t = \boldsymbol{s}_t', A_t = \boldsymbol{a}_t]. \tag{28}$$

Policy-based RL algorithms operate by calculating an estimator for the policy gradient. The most widely utilized form

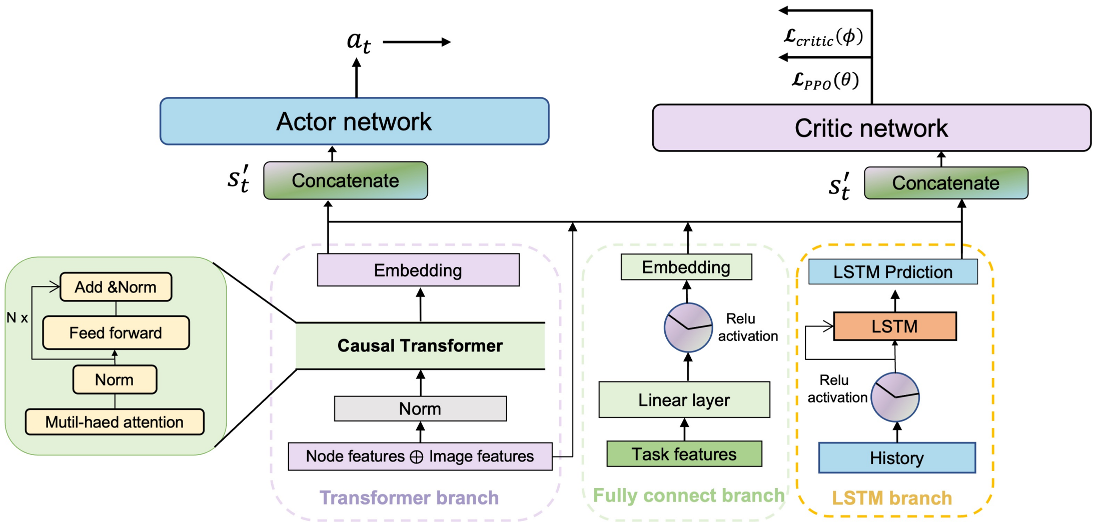


Fig. 3. Overview of the SATS algorithm. The SATS framework consists of an actor network and a critic network. The actor network incorporates a transformer branch, a linear neural network branch, and an LSTM branch. Essential operations of the algorithm encompass observing the system state, generating actions, calculating rewards, and updating the network.

of this gradient estimator is as follows:

$$\nabla \mathcal{L}_{PG}(\theta) = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t') \cdot \hat{A}(\boldsymbol{s}_t', \boldsymbol{a}_t) \right]. \tag{29}$$

In this context, π_θ denotes a stochastic policy, and $\hat{A}_t$ serves as an estimator for the advantage function at time t. However, employing $\mathcal{L}_{PG}(\theta)$ to optimize over several steps using the identical trajectory may lead to disproportionately large updates in the policy, which is typically not favorable. To mitigate this concern, the Generalized Advantage Estimator (GAE) [36] was introduced as a more refined estimator for advantages, calculated as follows:

$$\hat{A}_{\theta_{old}}(\boldsymbol{s}_t', \boldsymbol{a}_t) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \cdot \delta_{t+l}. \tag{30}$$

In this equation, λ is the GAE parameter that critically influences the balance between bias and variance, serving to decrease variance and enhance the speed of the learning process. The term

$$\delta_t = r_t + \gamma V_\phi(\boldsymbol{s}_{t+1}') - V_\phi(\boldsymbol{s}_t') \tag{31}$$

represents the Temporal Difference (TD) error at time step t, where $V_\phi(\boldsymbol{s}_t')$ denotes the value function approximated by a Deep Neural Network (DNN). Here, the critic network is updated by minimizing the loss defined as:

$$\mathcal{L}_{CRITIC}(\phi) = \hat{\mathbb{E}}_t \left[(V_\phi(\boldsymbol{s}_t') - \hat{V}_t)^2 \right], \tag{32}$$

where $\hat{V}_t$ can be obtained as $\hat{V}_t = \sum_{l=0}^{\infty} \gamma^l \cdot r_{t+l}$.

Furthermore, to avoid the pitfalls of local optima during training and sampling, we utilize Proximal Policy Optimization (PPO) [37]. PPO introduces a clipped surrogate objective in place of $\mathcal{L}_{PG}(\theta)$, characterized as

$$\begin{aligned}\mathcal{L}_{PPO}(\theta) = \hat{\mathbb{E}}_t \Big[\min \Big(& ra_t(\theta) \cdot \hat{A}_{\theta_{old}}(\boldsymbol{s}_t', \boldsymbol{a}_t), \\ & \text{clip}(ra_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \hat{A}_{\theta_{old}}(\boldsymbol{s}_t', \boldsymbol{a}_t) \Big) \Big],\end{aligned} \tag{33}$$

where $ra_t(\theta)$ is the policy probability radio defined as

$$ra_t(\theta) = \frac{\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t')}{\pi_{\theta_{old}}(\boldsymbol{a}_t|\boldsymbol{s}_t')} \tag{34}$$

and ϵ is a hyperparameter, e.g. $\epsilon = 0.1$. The function clip$(ra_t(\theta), 1-\epsilon, 1+\epsilon)$ is designed to constrain the value of $ra_t(\theta)$, which represents the ratio of the new policy to the old policy, within a specific range defined by $1-\epsilon$ and $1+\epsilon$. If $ra_t(\theta)$ falls below $1-\epsilon$, the function adjusts it to $1-\epsilon$; if $ra_t(\theta)$ exceeds $1+\epsilon$, it is reduced to $1+\epsilon$. Otherwise, if $ra_t(\theta)$ is already within the acceptable range, its value remains unchanged. This clipping mechanism moderates policy updates, preventing excessively large adjustments that could lead to instability in the learning process.

Attention-based Model. To tackle the dynamic and complex task scheduling challenges within the container-based VEC ecosystem, our model employs the Transformer architecture for embedding features of PVs and devices. This approach moves away from directly utilizing device features as state vectors, enabling a more refined representation of the VEC system state. Unlike Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN), the Transformer employs a unique self-attention mechanism, demonstrating stable and effective representation capabilities [38], [39]. This mechanism adeptly captures and analyzes intricate dependencies and interactions among the features of PVs' computing resources, a crucial aspect for effectively addressing the diverse scheduling challenges posed by varying computational resources and unique image storage configurations of PVs. In our model, we define query matrix $\boldsymbol{Q} \in \mathbb{R}^{M \times d_{model}}$, representing information extracted from the current system state to capture the immediate computational resource demands; $\boldsymbol{K} \in \mathbb{R}^{M \times d_{model}}$ be the key matrix, reflecting the historical data of PVs' computational resource characteristics; and $\boldsymbol{V} \in \mathbb{R}^{M \times d_{model}}$ be the value matrix, encoding the resource configuration information corresponding to the keys. Here, M is the number of PVs and devices, and d_{model} are the corresponding PVs and device state feature dimensions. The outputs of self-attention are

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_{model}}}\right)\boldsymbol{V}, \tag{35}$$

where the scalar $1/\sqrt{d_{model}}$ is applied to mitigate the risk of the softmax function entering regions with minimal gradients. Following this, we detail the multi-head attention mechanism as outlined below:

$$\begin{aligned}&\text{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \text{Concat}(\text{head}_1,\cdots,\text{head}_h)\boldsymbol{W}^O,\\&\text{head}_i = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q,\boldsymbol{K}\boldsymbol{W}_i^K,\boldsymbol{V}\boldsymbol{W}_i^V).\end{aligned} \tag{36}$$

Here, $\boldsymbol{W}^O$ represents the output weight matrix and h denotes the number of attention heads. Each head $head_i$ has its trainable weight matrices $\boldsymbol{W}_i^Q$, $\boldsymbol{W}_i^K$, and $\boldsymbol{W}_i^V$ for Q, K, and V, whose size is in $\mathbb{R}^{d_{model}\times(d_{model}/h)}$.

The position-wise feed-forward network (FFN) represents a fundamental component of the Transformer architecture, comprising two linear transformations with an intervening ReLU activation function. The dimensions of both the input and output are denoted by d_{model}, while the internal layer, characterized by the dimensionality d_{ff}, serves as a hyperparameter. Specifically, the FFN is defined as follows:

$$\text{FFN}(\boldsymbol{x}) = \max(0,\boldsymbol{x}\boldsymbol{W}_1+\boldsymbol{b}_1)\boldsymbol{W}_2+\boldsymbol{b}_2, \tag{37}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d_{model}\times d_{ff}}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{d_{ff}\times d_{model}}$ are the weights, and $\boldsymbol{b}_1 \in \mathbb{R}^{d_{ff}}$ and $\boldsymbol{b}_2 \in \mathbb{R}^{d_{model}}$ are the biases.

The same linear transformations are applied consistently across different positions within the sequence. It is important to highlight that position encoding is utilized to incorporate the sequence order information as follows:

$$\begin{aligned}&\text{PE}(pos,2i) = \sin(pos/10000^{2i/d_{model}}),\\&\text{PE}(pos,2i+1) = \cos(pos/10000^{2i/d_{model}}).\end{aligned} \tag{38}$$

Time sequence prediction. The LSTM branch is designed to strengthen the memory ability of the SATS algorithm since it allows the agent to incorporate a large amount of network measurement history into its state space to capture the long-term temporal dependencies of actual system dynamics. Specifically, the LSTM branch processes a sequence of historical task numbers as its input and predicts the task count for the upcoming time slot as its output. The LSTM branch is tailored to enrich information capture and enhance expressiveness for historical time slots, affording subsequent layers more immediate access to temporal data. When it comes to scheduling decisions, recognizing the temporal patterns of task arrivals is crucial, without being limited by the construction sequence of container images. This implies that scheduling should not depend on the sequential or adjacent relationships between images. LSTMs present a viable solution [40], [41]. By modeling interactions between pairs of features as the inner products of latent vectors, LSTMs can grasp complex temporal patterns without relying on a predefined sequence of container images. This attribute of LSTMs is particularly beneficial in scenarios where the timing and interplay of tasks are vital, yet not strictly bound to a specific operational sequence.

C. Computational Complexity Analysis

The time complexity analysis of the SATS algorithm, as presented in Algorithm 1, primarily encompasses four steps: system state processing, action selection, reward computation, and network updating. In the first step, we obtain the system state as defined in Eqn. (23). Since the state encompasses characteristics from M PVs, K_t tasks at time t, I images, and look-back steps of length $\mathcal{L}$, the time complexity for retrieving this state is calculated as $\mathcal{O}(7M+I\cdot M+6K_t+\mathcal{L})$. Second, the action is selected according to the policy network. The time complexity of this process depends on the network size and involves a forward pass with bounded complexity through three distinct components—Transformer, LSTM, and the three-layer fully connected network—each contributing a small cost, which can be considered constant and denoted as O_t [42]. Third, execute action $\boldsymbol{a}_t$ and receive reward r_t according to Eqn. (25). The complexity of reward calculation is $\mathcal{O}(K_t)$, as each task is assigned to a single device, and the computation for each task's time and energy is constant.

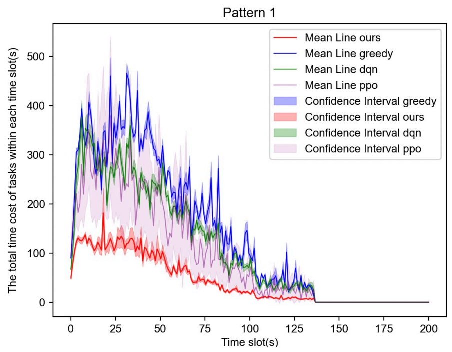
To evaluate the complexity of the network update, the distinct parts of the state are input into several neural network structures. Since the computational complexity at each time step is primarily influenced by the structure of these neural networks, the overall update complexity can be expressed as $\mathcal{O}(2(N_{\text{trans}}+N_{\text{fc}}+N_{\text{lstm}}+N_\pi+N_v))$. Here, $N_{\text{trans}} = \mathcal{O}(h_{tran}(Md_{model})^2 d_{model})$ accounts for the Transformer's self-attention mechanism with h_{tran} attention heads, sequence length Md_{model}, and embedding dimensionality d_{model}. $N_{\text{lstm}} = \mathcal{O}(\mathcal{L}(4n_h\mathcal{L}+4n_h^2))$ captures the LSTM layers' complexity, where $\mathcal{L}$ is the look back steps and n_h is the number of hidden units. $N_{\text{fc}} = \mathcal{O}(6K_t n_{fc})$ represents the complexity of the fully connected layers, which have a simple structure with 3 hidden layers, where n_{fc} is the number of neurons per hidden layer. Lastly, the computational complexity of the policy network, $N_\pi = \mathcal{O}((2(7M+I\cdot M)+6K_t+\mathcal{L})\cdot n_\pi)$, reflects its simple structure with two hidden layers, where n_π is the number of neurons per hidden layer. The value network has a similar structure, and its complexity is analogous to that of the policy network. In each episode, the algorithm collects experiences over $|\mathcal{T}|$ time steps and then performs training. The total complexity for experience collection is $\mathcal{O}(|\mathcal{T}|\times(7M+I\cdot M+7K_t+\mathcal{L})$. During the training phase, for K epochs, the computational complexity per epoch is: $\mathcal{O}(\mathcal{B}\times(N_{\text{trans}}+N_{\text{fc}}+N_{\text{lstm}}+2N_\pi+P_\theta))$ where $\mathcal{B}$ is the batch size, P_θ is the numbers of parameters in the networks. In this setting, most parameters remain constant, so the computational complexity of the algorithm is determined by the complexity of the training parameters, as well as the number of episodes E and the number of time steps $|\mathcal{T}|$ per episode.

V. Numerical Simulations

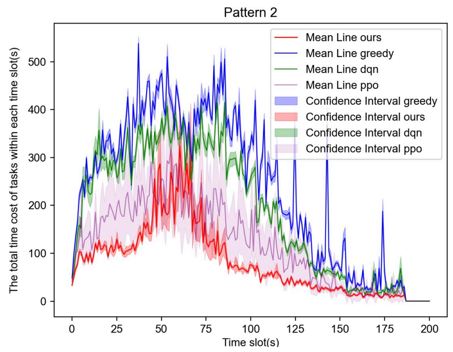
In this section, we evaluate the performance of SATS by outlining the experimental setup, followed by a presentation and analysis of the experimental findings.

A. Experimental Settings

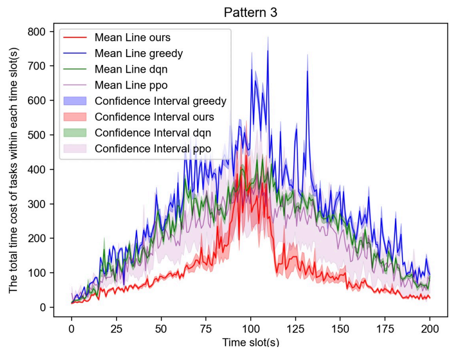
Parameter settings. To assess the effectiveness of our proposed algorithms, we implemented a simulation environ-



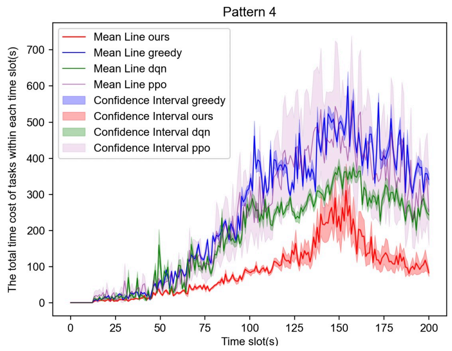
(a) Task Arrival Pattern 1



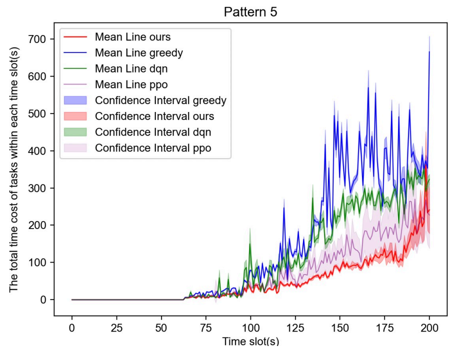
(b) Task Arrival Pattern 2



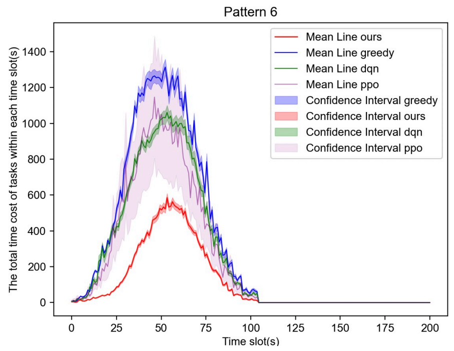
(c) Task Arrival Pattern 3



(d) Task Arrival Pattern 4



(e) Task Arrival Pattern 5



(f) Task Arrival Pattern 6

Fig. 4. Trend of total time cost per task in a single time slot across various task arrival patterns. This includes 6 PVs ($N = 6$) and 7 devices ($M = 7$), employing a weight factor of $w_t = 1$.

ment in Python. This environment simulates PVs entering and exiting a generic parking lot with 50 spaces, behaving in a random and dynamic fashion within a $100m \times 100m$ area. Each PV's storage capacity is randomly assigned within the range of 5 GB to 10 GB, and their available bandwidth varies from 60 Mbps to 90 Mbps. The CPU frequencies of the PVs are randomly selected to fall within the 0.8 GHz to 1.2 GHz range, with memory allocations for the PVs varying from 2 GB to 14 GB. This variation reflects the heterogeneity of the devices, which are distributed randomly across the simulation. Meanwhile, local devices feature CPU frequencies are set between 0.1 GHz and 0.5 GHz, with memory allocations ranging from 256 MB to 2 GB. The transmission power is set at 23 dBm, and the noise power spectral density is fixed at -174 dBm/Hz. Wireless communication is conducted via sensor nodes and a single edge node, with a transmission bandwidth spectrum ranging from 24 GHz to 39 GHz. Tasks are generated at random, with sizes varying from 10 KB to 10 MB.

Simulation of Task Arrival Patterns. Task arrival trends are modeled using a Gaussian distribution, characterized by predefined mean (μ) and standard deviation (σ) values. These parameters, chosen from five unique mean values and two standard deviation options, generate diverse task density patterns across a single episode of 200 time slots. For every episode, the simulation constructs a probability density function (PDF) from the designated mean (μ) and standard deviation (σ), depicting the distribution of task arrivals during that period. The resulting PDF values are scaled and adjusted by introducing a random noise factor between 0.8 and 1.2, enhancing the variability in task generation. This creates time slots with realistically distributed tasks, mirroring potential real-world scenarios. Each pattern equally contributes to the training dataset, ensuring uniform exposure in the simulation. After generation, the data are shuffled to evaluate the algorithm's robustness in a mixed data environment, providing a comprehensive test of its performance under varied conditions.

B. Baselines Description.

To benchmark the performance of our approach, we implement and evaluate several baseline algorithms.

- ***Greedy Selection (Greedy)***. It adopts a greedy strategy for task distribution to either PVs or local execution. It evaluates the available options (available PVs or local processing) based on key performance indicators such as CPU frequency, transmission power, computational capacity, and download delay. The algorithm determines a composite cost metric that encapsulates energy consumption and time expenditure, aiming to identify the most efficient PV or local device that offers rapid computation with minimal energy usage. This integrated assessment ensures that the selected node optimizes both speed and energy efficiency for task execution. Choices are ranked based on this metric, with the algorithm assigning tasks to the option that minimizes energy and time costs, thereby optimizing resource utilization.
- ***Deep-Q Network (DQN)*** [43]. It is a Deep RL algorithm that combines DNN with Q-learning to learn optimal

policies in large state and action spaces. DQN is based on the principle of value iteration, where an agent learns to approximate the optimal action-value function through interaction with the environment.

- ***Proximal Policy Optimization (PPO)*** [37]. It is an RL algorithm that aims to optimize policies efficiently and stably. It belongs to the class of actor-critic algorithms and is particularly known for its simplicity and effectiveness. PPO operates by iteratively updating policy parameters based on estimations of the expected return, using a clipped surrogate objective function to maintain a balance between exploration and exploitation.

C. Performance Across Diverse Temporal Patterns

Figure 4 illustrates the trend of total time cost per task in a single time slot across various task arrival patterns, incorporating scenarios with 6 parked vehicles (PVs) and 7 devices, and accounting for a weight factor (w_t) of 1. Tasks are generated across a sequence of 200 time slots per episode, and the plotted data correspond to the average time costs accrued over the final 1,000 episodes. This analysis emphasizes long-term performance to provide a thorough evaluation of the system's efficiency. The task arrivals are categorized into six distinct patterns. As illustrated in Figure 4(a) - 4(f), each corresponds to one of the task arrival patterns (Patterns 1 through 6). SATS demonstrates superior performance by consistently achieving lower total costs across most time slots and patterns compared to competing methods. This performance advantage highlights our algorithm's effectiveness in managing time-related costs across diverse task arrival patterns and operational conditions. The greedy algorithm, while competitive in certain slots, generally incurs a higher cost. The shaded regions represent the confidence intervals, indicating the variability and reliability of the mean cost values. In Pattern 1, as shown in Figure 4(a), SATS demonstrates significant time cost efficiency, particularly in the middle time slots. A similar trend is observed in Patterns 2 and 3, as illustrated in Figure 4(b) and 4(c), where SATS outperforms the alternatives, especially during periods of peak task arrivals. Figure 4(d) and 4(e), representing Patterns 4 and 5, showcase the robustness of SATS in handling fluctuating costs, with the confidence intervals for our method being narrower than those for other methods, suggesting greater stability. Finally, Figure 4(f) highlights Pattern 6, where SATS shows superior performance with a more pronounced difference from the other methods. This is particularly noteworthy during the latter time slots, where SATS appears to adapt more effectively to the changes in task patterns. Overall, SATS not only yields a lower mean cost but also displays consistency and reliability across various task arrival patterns, emphasizing its suitability for dynamic and uncertain environments.

D. Performance Analysis on Varying Numbers of PVs

Table II presents the performance metrics of three baseline algorithms compared with SATS. The comparison is conducted over different quantities of PVs, with a constant device number $M - N = 10$. Metrics include the average time cost per task, the energy cost per task, and the reward per task over $1,000$ episodes. These metrics are critical for assessing the effectiveness of SATS in a VEC network setup. Lower time and energy costs indicate improved performance, while a higher reward signifies a more optimal balance between time and energy expenditures.

TABLE II
PERFORMANCE METRICS FOR DIFFERENT METHODS GIVEN DIFFERENT NUMBERS OF PVS, WHERE DEVICES NUMBER = 10, PENALTY = 100, AND WEIGHT FACTOR $w_t = 0.5$

Algorithm	**PVs: 5**	**PVs: 10**	**PVs: 15**	**PVs: 20**
Time Cost per Task				
Greedy	515.4988	585.9003	561.4906	497.3194
DQN	191.0666	26.4591	194.5597	29.8914
PPO	148.8996	22.5706	45.3104	24.0789
Ours	**26.5569**	**20.3930**	**32.4720**	**19.9474**
Energy Cost per Task				
Greedy	1916.8043	1917.4010	1911.6688	1933.0821
DQN	7807.2273	7448.5881	3591.5511	7913.0113
PPO	7301.3018	8335.6894	4738.0762	8687.1297
Ours	9129.4580	5988.9614	10677.5660	5403.5835
Reward per Task				
Greedy	-863.804	-888.897	-857.785	-852.094
DQN	-284.118	-119.356	-288.951	-122.570
PPO	-241.902	-115.576	-137.975	-116.840
Ours	**-119.670**	**-113.287**	**-125.544**	**-111.979**

The greedy algorithm incurs the highest time cost per task, notably with 515.50 and 585.90 for 5 and 10 PVs, respectively. In stark contrast, SATS maintains the lowest time cost across varying PV numbers, indicative of superior efficiency in task management. The underlying issue with the greedy approach is its narrow focus on options that ostensibly offer the quickest completion and minimal energy usage, leading to a predisposition towards local devices deemed energy-efficient. Such a strategy neglects critical factors like download costs and device workload, which can substantially decelerate computation. As a result, even when an available PV could offer better performance than a local device, the greedy algorithm's disregard for these aspects compromises its decision-making efficacy. Conversely, SATS evaluates the time constraints of each task, adopting a more holistic approach to ensure optimal task distribution within the defined constraints, thereby optimizing overall efficiency.

For energy cost per task, SATS surpasses other approaches in efficiency, with the exception of the greedy method, due to the reasons previously discussed. This efficiency becomes particularly evident at higher PV counts, where our algorithm significantly reduces energy consumption, achieving its lowest at 5403.58 for 20 PVs. This trend emphasizes the inherent scalability and energy optimization of SATS, showcasing its ability to adapt to different network scales while effectively balancing energy consumption with time efficiency.

For reward per task, negative values serve as indicators of

TABLE III
BREAKDOWN OF TIME COST COMPARISON FOR DIFFERENT METHODS GIVEN DIFFERENT NUMBERS OF PVS, WHERE DEVICES NUMBER = 10 AND WEIGHT FACTOR $w_t = 0.5$.

Algorithm	**PVs: 5**	**PVs: 10**	**PVs: 15**	**PVs: 20**
Computation Time Cost per Task				
Greedy	534.13554	584.5138	531.2083	495.8514
DQN	157.9777	2.6284	176.4882	5.7115
PPO	126.8658	3.3917	30.6617	3.3863
Ours	**3.5621**	**2.7880**	**3.3978**	**2.0997**
Transmission Time Cost per Task				
Greedy	1.2420	1.3267	1.3077	1.4285
DQN	0.8654	1.4201	0.7152	1.1923
PPO	1.0532	0.8174	1.0917	0.8133
Ours	0.4096	1.3338	0.8215	1.2546
Download Time Cost per Task				
Greedy	0.0994	0.0599	0.0636	0.0396
DQN	32.2236	22.4106	19.2373	22.9877
PPO	20.9806	18.3615	13.5570	19.8793
Ours	22.5851	16.2711	28.2528	16.5930

achieving an optimal balance between time and energy expenditures, as previously elucidated. SATS consistently exhibits the maximum values, underscoring its superior capacity to optimize computational tasks in terms of energy and time efficiency, affirming its comprehensive effectiveness.

E. Time Cost on Different Numbers of PVs

Table III provides an in-depth analysis of time costs associated with baselines, given a fixed number of devices ($N - M = 10$) and a weight factor of $w_t = 0.5$, across varying numbers of PVs. By breaking down the overall time cost into three distinct components: Computation time, download time, and transmission time, the table effectively illustrates the efficiency and optimization aspects of each method in a granular manner.

In terms of computation time, SATS consistently outperforms competitors across all PV configurations, showcasing exceptional efficiency. Notably, at the configuration with 10 PVs, the computation time of SATS is significantly lower—nearly one-eighth of the next-best competitor, DQN. This substantial reduction in computation time is attributed to SATS's ability to effectively assess and utilize the computing resources of different PVs, which allows it to make more informed offloading polices that harmoniously balance image download demands with computational capabilities. Furthermore, the transmission time results reveal a strategic choice within the SATS algorithm to not prioritize minimal transmission times. Instead, it opts for a deliberate trade-off, enhancing overall time cost effectiveness. By accepting longer initial transmission periods, SATS strategically focuses on reducing computation time through more precise task offloading. This approach underscores a nuanced understanding of PVs' computational resource variations and leverages more adeptly than competing methods like PPO and DQN. Additionally, the average performance of STAS in download time illustrates its capability to achieve shorter computation times by striking a balance in the downloading of container sequences. This method significantly boosts the efficiency of task downloads, demonstrating an optimized management of both download and computation times. By achieving this balance, SATS not only reduces total operational time but also ensures that task downloads are managed in the most efficient way possible, resulting in significant enhancements in overall system performance. This strategy enhances the efficiency of computational resources and container-based scheduling, showcasing the capability of STAS to streamline network operations effectively.

Across all metrics, SATS surpasses competing strategies, underscoring its robustness and superior performance in managing tasks within a vehicular network environment. The results highlight the capacity of STAS to significantly improve operational efficiency in practical applications, marking a substantial contribution to advancements in vehicular computational frameworks. This effectiveness not only demonstrates its technical excellence but also its potential applicability in enhancing real-world vehicular systems.

VI. CONCLUSION

In this paper, we presented the SATS algorithm, a policy gradient-based deep RL approach designed to optimize task offloading in a VEC scenario that leverages parked vehicles as computational nodes. Integrating parked vehicles with container technology extends the existing MEC infrastructure and enhances load balancing, auto-healing, resource isolation, and security features. To capture the complex structure of computing nodes and patterns in time-series tasks, such as peak hours and congestion, SATS employs Transformer and LSTM to enhance task scheduling precision and overall performance. The experimental evaluations, conducted through numerous numerical comparisons with three baseline algorithms, demonstrate the significant advantages of SATS in reducing delays and energy consumption across the network.

In the future, several directions could extend this work. First, vehicle modeling can be improved by incorporating sudden vehicle departures and analyzing daily parking behavior records, such as recording and predicting the frequency and duration of parking, peak parking times, and the number of vehicles entering and leaving different locations to improve model robustness. Second, we plan to use real datasets (e.g., real task arrival sequences, practical PV energy costs, and real-world computational tasks) to make our experimental results more convincing. Third, we aim to extend our work to multiple base stations, covering larger areas, which will allow us to evaluate the model's scalability and distributed nature in highly dynamic IoV environments.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 62202055, the National Key R&D Program of

China under Grant No. 2022YFE0201400, the Start-up Fund from Beijing Normal University under Grant No. 310432104/312200502510, the Internal Fund from BNU-HKBU United International College under Grant No. UICR0400003-24, the Project of Young Innovative Talents of Guangdong Education Department under Grant No. 2022KQNCX102, and the Interdisciplinary Intelligence Super-Computer Center, Beijing Normal University (Zhuhai).

References

[1] X. Huang, R. Yu, S. Xie, and Y. Zhang, "Task-container matching game for computation offloading in vehicular edge computing and networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 10, pp. 6242–6255, 2020.

[2] K. Nguyen, S. Drew, C. Huang, and J. Zhou, "Collaborative container-based parked vehicle edge computing framework for online task offloading," in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. IEEE, 2020, pp. 1–6.

[3] X. Huang, P. Li, and R. Yu, "Social welfare maximization in container-based task scheduling for parked vehicle edge computing," *IEEE Communications Letters*, vol. 23, no. 8, pp. 1347–1351, 2019.

[4] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11 158–11 168, 2019.

[5] T. Lu, F. Zeng, J. Shen, G. Chen, W. Shu, and W. Zhang, "A scheduling scheme in a container-based edge computing environment using deep reinforcement learning approach," in *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, 2021, pp. 56–65.

[6] W. Fan, M. Hua, Y. Zhang, Y. Su, X. Li, B. Tang, F. Wu, and Y. Liu, "Game-based task offloading and resource allocation for vehicular edge computing with edge-edge cooperation," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 6, pp. 7857–7870, 2023.

[7] X. Huang, R. Yu, D. Ye, L. Shu, and S. Xie, "Efficient workload allocation and user-centric utility maximization for task scheduling in collaborative vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3773–3787, 2021.

[8] D. Han, W. Chen, and Y. Fang, "A dynamic pricing strategy for vehicle assisted mobile edge computing systems," *IEEE Wireless Communications Letters*, vol. 8, no. 2, pp. 420–423, 2018.

[9] Y. Zhang, C.-Y. Wang, and H.-Y. Wei, "Parking reservation auction for parked vehicle assistance in vehicular fog computing," *IEEE transactions on vehicular technology*, vol. 68, no. 4, pp. 3126–3139, 2019.

[10] X. Huang, P. Li, R. Yu, Y. Wu, K. Xie, and S. Xie, "Fedparking: A federated learning based parking space estimation with parked vehicle assisted edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 9355–9368, 2021.

[11] Y. Ren, X. Chen, S. Guo, S. Guo, and A. Xiong, "Blockchain-based vec network trust management: A drl algorithm for vehicular service offloading and migration," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 8148–8160, 2021.

[12] S.-s. Lee and S. Lee, "Resource allocation for vehicular fog computing using reinforcement learning combined with heuristic information," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 450–10 464, 2020.

[13] Z. Zou, X. Yan, Y. Yuan, Z. You, and L. Chen, "Attention mechanism enhanced lstm networks for latency prediction in deterministic mec networks," *Intelligent Systems with Applications*, vol. 23, p. 200425, 2024.

[14] T.-V. Nguyen, N.-N. Dao, W. Noh, S. Cho *et al.*, "User-aware and flexible proactive caching using lstm and ensemble learning in iot-mec networks," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3251–3269, 2021.

[15] L. Xu, M. Qin, Q. Yang, and K.-S. Kwak, "Learning-aided dynamic access control in mec-enabled green iot networks: A convolutional reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 2, pp. 2098–2109, 2022.

[16] Z. H. Meybodi, A. Mohammadi, M. Hou, E. Rahimian, S. Heidarian, J. Abouei, and K. N. Plataniotis, "Multi-content time-series popularity prediction with multiple-model transformers in mec networks," *Ad Hoc Networks*, vol. 157, p. 103436, 2024.

[17] X. Zhang, J. Zhang, Z. Liu, Q. Cui, X. Tao, and S. Wang, "Mdp-based task offloading for vehicular edge computing under certain and uncertain transition probabilities," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3296–3309, 2020.

[18] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11 158–11 168, 2019.

[19] J. Feng, Z. Liu, C. Wu, and Y. Ji, "Ave: Autonomous vehicular edge computing framework with aco-based scheduling," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 12, pp. 10 660–10 675, 2017.

[20] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, 2016.

[21] W. Fan, J. Liu, M. Hua, F. Wu, and Y. Liu, "Joint task offloading and resource allocation for multi-access edge computing assisted by parked and moving vehicles," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 5, pp. 5314–5330, 2022.

[22] S. Ge, M. Cheng, X. He, and X. Zhou, "A two-stage service migration algorithm in parked vehicle edge computing for internet of things," *Sensors*, vol. 20, no. 10, p. 2786, 2020.

[23] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE wireless communications*, vol. 24, no. 3, pp. 48–56, 2017.

[24] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[25] A. Ahmed and G. Pierre, "Docker container deployment in fog computing infrastructures," in *2018 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2018, pp. 1–8.

[26] S. Hoque, M. S. De Brito, A. Willner, O. Keil, and T. Magedanz, "Towards container orchestration in fog computing infrastructures," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2017, pp. 294–299.

[27] Z. Tang, F. Mou, J. Lou, W. Jia, Y. Wu, and W. Zhao, "Joint resource overbooking and container scheduling in edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–15, 2024.

[28] W. Feng, N. Zhang, S. Li, S. Lin, R. Ning, S. Yang, and Y. Gao, "Latency minimization of reverse offloading in vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 5, pp. 5343–5357, 2022.

[29] T. Lu, F. Zeng, J. Shen, G. Chen, W. Shu, and W. Zhang, "A scheduling scheme in a container-based edge computing environment using deep reinforcement learning approach," in *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*. IEEE, 2021, pp. 56–65.

[30] M. Wu, W. Qi, J. Park, P. Lin, L. Guo, and I. Lee, "Residual energy maximization for wireless powered mobile edge computing systems with mixed-offloading," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 4, pp. 4523–4528, 2022.

[31] X. Wang, J. Li, Z. Ning, Q. Song, L. Guo, S. Guo, and M. S. Obaidat, "Wireless powered mobile edge computing networks: A survey," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–37, 2023.

[32] M. B. Majed, T. A. Rahman, and O. A. Aziz, "Propagation path loss modeling and outdoor coverage measurements review in millimeter wave bands for 5g cellular communications," *International Journal of Electrical and Computer Engineering*, vol. 8, no. 4, p. 2254, 2018.

[33] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. Lau, "Dynamic virtual machine management via approximate markov decision process," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[34] A. Sagheer and M. Kotb, "Time series forecasting of petroleum production using deep lstm recurrent networks," *Neurocomputing*, vol. 323, pp. 203–213, 2019.

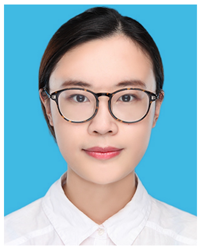
[35] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[36] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[39] L. Meng, M. Wen, Y. Yang, C. Le, X. Li, W. Zhang, Y. Wen, H. Zhang, J. Wang, and B. Xu, "Offline pre-trained multi-agent decision transformer: One big sequence model tackles all smac tasks," *arXiv preprint arXiv:2112.02845*, 2021.
[40] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.
[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[42] Z. Tang, J. Lou, and W. Jia, "Layer dependency-aware learning scheduling algorithms for containers in mobile edge computing," *IEEE Transactions on Mobile Computing*, 2022.
[43] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.


Jianqiu Wu received the M.S. degree from the Faculty of Engineering, the Chinese University of Hong Kong, in 2018. She is currently pursuing an M.Phil. degree with the Department of Computer Science, BNU-HKBU United International College, Zhuhai, China. She is supervised by Dr. Jianxiong Guo, and her research interests include reinforcement learning, mobile edge computing, and deep learning in wireless communications.

Jianxiong Guo (Member, IEEE) received his Ph.D. degree from the Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA, in 2021, and his B.E. degree from the School of Chemistry and Chemical Engineering, South China University of Technology, Guangzhou, China, in 2015. He is currently an Associate Professor with the Advanced Institute of Natural Sciences, Beijing Normal University, and also with the Guangdong Key Lab of AI and Multi-Modal Data Processing, BNU-HKBU United International College, Zhuhai, China. He is a member of IEEE/ACM/CCF. He has published more than 80 peer-reviewed papers and been the reviewer for many famous international journals/conferences. His research interests include social networks, wireless sensor networks, combinatorial optimization, and machine learning.

Zhiqing Tang (Member, IEEE) received the B.S. degree from the School of Communication and Information Engineering, University of Electronic Science and Technology of China, China, in 2015 and the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2022. He is currently an assistant professor with the Advanced Institute of Natural Sciences, Beijing Normal University, China. His current research interests include edge computing, resource scheduling, and reinforcement learning.

Chuanwen Luo received his PhD degree in 2020 from the School of Information, Renmin University of China, Beijing, China. Currently, he is working as an Assistant Professor at the School of Information Science and Technology, Beijing Forestry University, Beijing, China. He was a visiting scholar at the Department of Computer Science of the University of Texas at Dallas in 2019. His research interests include various topics in the application of wireless networks, ad hoc & sensor networks, algorithm design and analysis, etc.

Tian Wang (Senoir Member, IEEE) received his BSc and MSc degrees in Computer Science from the Central South University in 2004 and 2007, respectively. He received his PhD degree from the City University of Hong Kong in Computer Science in 2011. Currently, he is a professor with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University. His research interests include the Internet of Things, Edge Computing, and Mobile Computing. He has more than 15000 citations, according to Google Scholar. His H-index is 71.

Weijia Jia (Fellow, IEEE) is currently a Chair Professor, Director of BNU-UIC Institute of Artificial Intelligence and Future Networks, Beijing Normal University (Zhuhai) and VP for Research of BNU-HKBU United International College (UIC) and has been the Zhiyuan Chair Professor of Shanghai Jiao Tong University, China. He was the Chair Professor and the Deputy Director of the State Kay Laboratory of Internet of Things for Smart City at the University of Macau. He received BSc/MSc from Center South University, China in 82/84 and Master of Applied Sci./PhD from Polytechnic Faculty of Mons, Belgium in 92/93, respectively, all in computer science. From 93-95, he joined German National Research Center for Information Science (GMD) in Bonn (St. Augustine) as a research fellow. From 95-13, he worked at the City University of Hong Kong as a professor. His contributions have been recognized as optimal network routing and deployment; anycast and QoS routing, sensors networking, AI (knowledge relation extractions; NLP, etc.), and edge computing. He has over 600 publications in the prestige international journals/conferences and research books and book chapters. He has received the best product awards from the International Science & Tech. Expo (Shenzhen) in 20112012 and the 1st Prize of Scientific Research Awards from the Ministry of Education of China in 2017 (list 2). He has served as area editor for various prestige international journals, chair, PC member, and keynote speaker for many top international conferences. He is the Fellow of IEEE and the Distinguished Member of CCF.