A DRL-based Load-Balanced Task Offloading Approach for Vehicular Edge Computing

Shucai Wang[†], Chaogang Tang[†], Shuo Xiao[†], Haifeng Jiang[†], Huaming Wu[‡], Ruidong Li[§]
[†]School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, Jiangsu, China
[‡]Center for Applied Mathematics, Tianjin University, 300072, Tianjin, China
[§]Institute of Science and Engineering, Kanazawa University, Kanazawa 920-1192, Japan
Email: {shucai, cgtang, sxiao, jhfeng}@cumt.edu.cn, whming@tju.edu.cn, liruidong@ieee.org

***Abstract*—The Vehicular Edge Computing (VEC) paradigm significantly reduces task processing latency in Internet of Vehicles (IoV) and Intelligent Transportation Systems (ITS) by deploying computational resources at Roadside Units (RSUs). However, the high mobility of vehicles and dynamic task arrivals lead to uneven load distribution among RSUs, severely impacting system performance. Actually, load balancing as an important evaluation metric for VEC system greatly affects the performance of individual edge servers in terms of latency, energy consumption, and task completion rates. In view of this, we propose a Proximal Policy Optimization (PPO) based deep reinforcement learning (DRL) approach to determine the task offloading and migration decisions and incorporate the fairness into the constraint, aiming to achieve efficient load-balanced task offloading in VEC. Particularly, we introduce a metric named Load Balancing Metric (LBM) to optimize RSU resource allocation and employ dynamical task migration strategies to optimize the metric. Simulation results demonstrate that this approach significantly enhances load balancing performance, reduces average latency and energy consumption, and provides an efficient resource scheduling solution for VEC systems.**

***Index Terms*—load balancing, deep reinforcement learning, migration, offloading, VEC.**

I. Introduction

With the rapid development of the Internet of Vehicles (IoV) and Intelligent Transportation Systems (ITS), vehicular tasks, such as autonomous driving decision-making and road condition perception, are increasingly characterized by computationally intensive processing requirements and stringent time constraints [1]. Traditional cloud computing architectures face substantial challenges due to long-distance data transmission, resulting in high latency [2], and intrinsic bandwidth limitations, rendering them insufficient for real-time demands. Hence, Vehicular Edge Computing (VEC) has been put forward as a promising paradigm to tackle these issues. In VEC, computing servers are deployed on Roadside Units (RSUs) to bring computing resources closer to data sources, thereby significantly reducing task processing delays [3]. However, the promise of VEC is challenged by a complex interplay of factors: the high mobility of vehicles [4], the finite resources of RSUs, and the stochastic nature of task arrivals. These elements collectively create a highly dynamic environment where load imbalances among RSUs are not an exception, but the norm, leading to processing bottlenecks and degraded Quality of Service (QoS) [5]–[7].

While extensive research has sought to optimize VEC systems, most studies address these challenges in a fragmented manner, revealing a significant research gap. One stream of work prioritizes minimizing latency and energy consumption through sophisticated task offloading strategies [8]–[10]. However, they often overlook the critical issue of load distribution, potentially leading to scenarios where some RSUs are overloaded while others remain idle. Another stream of research does consider load balancing, but frequently relies on static system models or heuristic rules that lack adaptability to the network's rapid dynamics [11]. Crucially, many existing approaches neglect the necessity of proactive task migration—the process of relocating an already-offloaded task to a new RSU as the vehicle moves. Even among recent DRL based approaches [12]–[15], many focus solely on the initial offloading decision, failing to provide a holistic policy that can react to mobility-induced changes. This reveals a critical need for a unified and adaptive framework that can jointly optimize initial task offloading and subsequent task migration in response to real-time network dynamics, while explicitly enforcing load fairness across RSUs.

To bridge this gap, we propose a novel framework that leverages deep reinforcement learning to jointly optimize task offloading, migration, and resource allocation in dynamic VEC environments. By formulating the problem as a minimization of a weighted sum of latency and energy consumption under a dynamic load-balancing constraint, we aim to provide a robust and efficient scheduling solution. The primary contributions of this paper are threefold:

- We introduce a dynamic load balancing mechanism that adapts to real-time fluctuations in network-wide task loads. This mechanism is integrated into a PPO based DRL framework, which learns to make adaptive decisions on both initial task offloading and subsequent migration. This approach effectively mitigates RSU overload during peak traffic while minimizing unnecessary migration overhead, outperforming static balancing strategies.
- We formulate a joint optimization problem that minimizes system latency and energy consumption while explicitly incorporating fairness among RSUs. Specifically, we de-

fine a Load Balancing Metric (LBM) and, more importantly, introduce a dynamic load balancing threshold. This threshold adjusts the strictness of the fairness constraint based on the overall system task volume, allowing for greater flexibility during high-load periods and improved resource utilization during off-peak times.
- We conduct extensive simulation experiments to validate the effectiveness and robustness of our proposed approach. Compared to several key baselines, including a standard Actor-Critic (AC) algorithm and greedy strategies, our approach demonstrates significant improvements. The results show superior performance in terms of convergence speed, final objective value, average task response time, and the ability to maintain load balance across RSUs under varying task loads.

The remainder of this paper is organized as follows. Section II reviews the related research on task offloading and load balancing in vehicular edge computing. Section III establishes the system model, describing in detail the interaction mechanisms among vehicles, RSUs, and the cloud center. Section IV presents the formal modeling of the optimization problem and the corresponding algorithm design. Section V conducts simulation experiments to compare the proposed method with several benchmark algorithms, thereby validating its effectiveness and superiority. Finally, Section VI concludes the paper and discusses potential directions for future research.

II. Related Work

In mobile edge computing (MEC) and VEC environments, classical studies on task offloading and resource allocation mainly rely on optimization modeling and heuristic algorithms. Shen et al. [16] formulated task-dependent offloading and service caching as a Mixed Integer Nonlinear Programming (MINLP) problem and used semi-distributed dynamic programming to enhance offloading efficiency under a latency–energy weighted objective. Cang et al. [13] considered an OFDMA-based MEC system and maximized the minimum computational efficiency by alternately solving continuous and discrete subproblems, achieving joint optimization of power, computation, and subchannel allocation while improving fairness. From a multi-objective perspective, Luo et al. [17] proposed the PSOCO framework based on particle swarm optimization (PSO) to jointly minimize delay and cost in vehicular edge computing, revealing the Pareto trade-off between them. Building on these studies, Cheng et al. [18] further formulated the joint offloading and caching problem in VEC as NP-hard and introduced a collaborative optimization mechanism combining dynamic programming and a many-to-one matching game, marking an evolution from single-resource to multi-resource and multi-objective modeling.

With increasingly dynamic and uncertain networks, DRL has been adopted for online and scalable task offloading. Huang et al. [11] proposed the DROO framework integrating binary offloading and wireless resource allocation into a DRL paradigm, achieving near-optimal performance under fast-fading channels. Yang and Liu et al. [10] combined

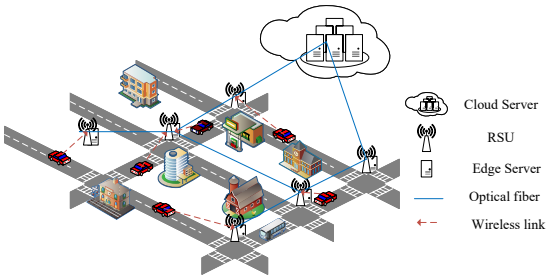

Fig. 1. System model.

asynchronous federated learning with DRL for location- and orientation-aware content prefetching and caching, effectively reducing request latency. Cao et al. [19] addressed peak and off-peak periods in VEC by combining fuzzy inference with reinforcement learning to balance resource utilization and QoS. Ahmed et al. [20] extended DRL to multi-hop offloading via the MVTO framework based on PPO, leveraging V2V and 5G V2X communication to reduce latency and enhance robustness in highly mobile networks. Collectively, these works indicate a shift from static optimization toward a closed-loop "perception–decision–learning" paradigm for task offloading under complex and time-varying conditions.

At the system level, load balancing has become critical to maintaining availability and cost efficiency at the network edge. Wu et al. [21] unified task offloading and load balancing in VEC through a Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm and a Technique for Order Preference by Similarity to Ideal Solution (TOPSIS)-based collaboration mechanism, minimizing system cost and improving resource utilization. Jiang et al. [4] proposed the ALCoD framework for IoT edge container clusters, employing a "worst-fit decreasing" deployment strategy and DRL-based migration optimization to enhance response time, bandwidth utilization, and load balance. Zhu et al. [22] transformed the migration problem into a weighted tripartite graph matching model solved via the Kuhn–Munkres (KM) algorithm for adaptive migration under real-time load states. Liu et al. [23] introduced a cost-aware switch migration method from the control-plane perspective, improving controller load balancing and latency in edge–cloud architectures. Overall, load balancing research converges toward hybrid paradigms that integrate graph-matching or combinatorial optimization with learning-driven migration strategies, validated in both VEC and edge–cloud containerized systems.

III. System Model

The system model comprises three distinct entities: moving vehicles, RSUs, and a cloud center, as illustrated in Fig. 1. RSUs are connected to the cloud center via wired links, while vehicles communicate wirelessly with RSUs using Vehicle-to-Infrastructure (V2I) communication technology. When a vehicle generates a task, it sends an offloading request to the cloud center through the nearest RSU. The cloud center then determines whether the task should be offloaded for

computation. If the task offloading decision is made, the cloud center will designate the initial RSU for task offloading and the target RSU for task execution. Table I summarizes the key notations employed in the system model and subsequent analysis.

The VEC system under consideration comprises N vehicles and M RSUs. The set of vehicles is represented by $\mathcal{N} = \{1, 2, \cdots, N\}$, where each vehicle $n \in \mathcal{N}$ is characterized by signal transmission power p_n and CPU frequency f_n. To model a continuous workload, we consider a system where each vehicle generates a computationally intensive task at the beginning of each time slot $t \in \mathcal{T} = \{0, 1, \cdots, T-1\}$. Similarly, the set of RSUs is represented by $\mathcal{M} = \{1, 2, \cdots, M\}$, where each RSU $m \in \mathcal{M}$ has multiple parameters including CPU frequency F_m, total bandwidth B_m, and the maximum communication range R_m. Tasks generated by vehicles in each time slot t are denoted by the set $T(t) = \{T_1(t), T_2(t), \cdots, T_N(t)\}$, where each task $T_n(t)$ is defined as a triple $T_n(t) = \{C_n(t), D_n(t), T_n^{\max}(t)\}$, where $C_n(t)$ denotes the computational resources required to complete the task, $D_n(t)$ is the data volume, and $T_n^{\max}(t)$ represents the maximum allowable delay. A binary decision variable $x_{n,m}(t)$ is introduced to determine whether a task $T_n(t)$ is offloaded to RSU m. Particularly, $x_{n,m}(t) = 1$ indicates that the task is offloaded to RSU m, and $x_{n,m}(t) = 0$ indicates that the task is not offloaded to m. Additionally, $\sum_{m \in \mathcal{M}} x_{n,m}(t) = 0$ means that the task is executed locally at vehicle n.

TABLE I
KEY SYMBOLS AND DESCRIPTIONS

Symbol	Description
$\mathcal{N}$	Set of vehicles
$\mathcal{M}$	Set of RSUs
$\mathcal{T}$	Set of time slots
$T_n(t)$	Task generated by vehicle n at time t
$D_n(t)$	Data size of task $T_n(t)$
$C_n(t)$	Computation requirement of task $T_n(t)$
$T_n^{\max}(t)$	Maximum delay allowed for $T_n(t)$
p_n	Transmission power of vehicle n
f_n	CPU frequency of vehicle n
$h_{n,m}$	Channel gain between vehicle n and RSU m
$x_{n,m}(t)$	Offloading indicator
$y_{n,m'}(t)$	Migration indicator
$k_{m,m'}$	Hop count between RSU m and m'

A. Communication Model

The communication model determines the transmission rate between vehicles and RSUs, ensuring efficient data transmission and task migration among vehicles and RSUs. The transmission rate from vehicle n to RSU m, can be mathematically given as:

$$r_{n,m}(t) = \frac{B_m}{\sum_{i \in \mathcal{N}} x_{i,m}(t)} \cdot \log_2 \left(1 + \frac{p_n \cdot h_{n,m}}{\sigma^2}\right) \quad (1)$$

where $h_{n,m}$ denotes the channel gain between vehicle n and RSU m. σ^2 refers to the noise power. The term $\sum_{i \in \mathcal{N}} x_{i,m}(t)$ represents the total number of vehicles offloading their tasks to RSU m. The communication model employs a fair-sharing principle, where the bandwidth of RSU m is evenly distributed among all vehicles offloading to it [16].

B. Migration Model

The migration model is designed to manage those tasks that require migration between adjacent RSUs. Uneven workload distribution among RSUs can degrade the efficiency of the VEC system, while a task migration mechanism can efficiently facilitate load balancing across RSUs. Migrating tasks to suitable RSUs ensures successful task execution, and improvs the task completion rate. In the migration model, the migration variable $y_{n,m'}(t)$ is introduced to represent the migration status of task $T_n(t)$. Specifically, $y_{n,m'}(t) = 0$ indicates that the task is not migrated to RSU m'; $y_{n,m'}(t) = 1$ means that the task is ultimately executed at RSU m'. The RSUs are interconnected via high-speed optical fiber links, which provide a stable and high-capacity backhaul for task migration. The transmission rate between adjacent RSUs is thus considered a constant, denoted by R_0. Accordingly, the time taken for migration is given by

$$t_n^{mig}(t) = y_{n,m'}(t) \cdot k_{m,m'} \cdot \frac{D_n(t)}{R_0} \quad (2)$$

where $k_{m,m'}$ denotes the minimum number of hops between RSU m and RSU m'. If $m = m'$, it indicates that there is no task migration, and $k_{m,m'} = 0$. Additionally, the energy consumption for task migration is given by

$$e_n^{mig}(t) = y_{n,m'} \cdot k_{m,m'} \cdot p_{optical} \cdot t_n^{mig}(t) \quad (3)$$

where $p_{optical}$ represents the fixed power consumption of the optical fiber network interface device.

C. Task Computation Model

Vehicular tasks can be executed in two primary modes. Firstly, a task can be processed locally if the vehicle possesses adequate computing resources or if the task does not have stringent time constraints. Secondly, vehicular tasks can also be offloaded to RSUs for execution, if the vehicles seek ultra-low response latency. $\sum_{m \in \mathcal{M}} x_{n,m}(t) = 0$ denotes that the task $T_n(t)$ is executed locally on the vehicle n, with an execution time of

$$t_n^{local}(t) = \frac{C_n(t)}{f_n}. \quad (4)$$

Additionally, the energy consumption for local execution is given by

$$e_n^{local}(t) = \kappa_n \cdot (f_n)^2 \cdot C_n(t) \quad (5)$$

where κ_n is the energy consumption coefficient of the vehicle n.

If the task generated by vehicle n is offloaded to an RSU m for execution, the offloading transmission time for task $T_n(t)$ to RSU m is

$$t_n^{tra}(t) = x_{n,m}(t) \cdot \frac{D_n(t)}{r_{n,m}(t)} \quad (6)$$

and the computation time for task $T_n(t)$ at RSU m' is

$$t_n^{edge}(t) = y_{n,m'}(t) \cdot \frac{C_n(t)}{F_{n,m'}(t)} \tag{7}$$

where $F_{n,m'}(t)$ denotes the computational resources allocated by RSU m' to the task $T_n(t)$. The total energy consumption for task offloading at RSU m and execution at RSU m' includes offloading transmission energy $e_n^{tra}(t) = x_{n,m}(t) \cdot p_n \cdot t_n^{tra}(t)$, migration energy $t_n^{mig}(t)$, and computation energy $e_{n(t)}^{edge} = y_{n,m'} \cdot \kappa_m \cdot (F_{n,m'}(t))^2 \cdot C_n(t)$, where κ_m denotes the energy consumption coefficient of RSU m. Optimizing these parameters can minimize energy usage in scenarios with limited vehicle resources or high computational demands. Consequently, the time required for task offloading can be defined as $t_n^{rsu}(t) = t_n^{tra}(t) + t_n^{mig}(t) + t_n^{edge}(t)$, and the energy consumption is given as $e_n^{rsu}(t) = e_n^{tra}(t) + e_n^{mig}(t) + e_n^{edge}(t)$.

D. RSU Load Balance

It is noticeable that the resources of RSUs are not rich compared to the cloud center. Specifically, certain RSUs are equipped with powerful edge servers, granting them more substantial computing resources, whereas others have considerably fewer resources. Given the high mobility of vehicles, task arrivals at individual RSUs exhibit significant fluctuations, resulting in some RSUs experiencing overload while others remain underutilized with idle computing resources. Given this context, it is imperative to design load-balanced task offloading strategies in VEC. In this paper, we define the workload of RSU m as

$$\mathcal{L}_m(t) = \frac{L_m^{cpu}(t)}{F_m} \tag{8}$$

where $L_m^{cpu}(t) = \sum_{j \in \mathcal{N}} F_{j,m}(t)$. Here, $F_{j,m}(t)$ denotes the CPU resources allocated by RSU m to vehicle j, and $L_m^{cpu}(t)$ represents the total CPU resources already allocated by RSU m. The workload $\mathcal{L}_m(t)$ can intuitively reflect the resource utilization of RSU m.

We introduce LBM to comprehensively evaluate the load balancing of all RSUs in the VEC system. Specifically, the LBM is defined as the root mean square deviation of the RSU loads and given as

$$LBM(t) = \sqrt{\frac{1}{M} \sum_{m=1}^{M} \left(\mathcal{L}_m(t) - \bar{L}(t)\right)^2} \tag{9}$$

where $\bar{L}(t) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}_m(t)$. A smaller LBM value suggests that the loads across all RSUs are more evenly distributed around the average load, indicating a higher level of load balancing within the system. Conversely, a larger LBM value indicates substantial load disparities among RSUs, where certain RSUs are overloaded while others remain underutilized. Such imbalances may result in performance degradation, elevated task latency, and reduced system efficiency.

In VEC systems, vehicle-generated task data fluctuates significantly, making traditional static load balancing strategies inefficient. To address this, we introduce an adaptive load balancing mechanism. This mechanism relies on a dynamic upper bound for $LBM(t)$, which adjusts based on real-time system workload. This bound consists of two components: a static base threshold, denoted by δ, which represents a predefined tolerance for load imbalance, and a dynamic adjustment factor, $\varphi(t)$. The dynamic factor is defined as:

$$\varphi(t) = \alpha \cdot \left(\frac{\sum_{n \in \mathcal{N}} C_n(t)}{\sum_{n \in \mathcal{N}} C_n(t-1)} - 1 \right) \tag{10}$$

where $\sum_{n \in \mathcal{N}} C_n(t)$ is the total task computation amount at time slot t, and α is a sensitivity scaling factor. In essence, this combined threshold $\delta + \varphi(t)$ allows the system to relax the fairness constraint (larger bound) during high-load periods to accommodate processing pressure, and tighten it (smaller bound) during low-load periods to optimize resource utilization. This ensures efficient resource distribution and enhances overall system performance in dynamic vehicular environments.

IV. Problem Formulation and Algorithm Design

A. Optimization Objective

Based on the aforementioned description, the total time for all N vehicles across T time slots is calculated as:

$$T^{total} = \sum_{t=0}^{T-1} \sum_{n=1}^{N} \Big[(1 - \sum_{m \in \mathcal{M}} x_{n,m}(t)) \cdot t_n^{local}(t) + x_{n,m}(t) \cdot t_n^{rsu}(t) \Big] \tag{11}$$

and the average task time is:

$$T^{avg} = \frac{T^{total}}{N \cdot T} \tag{12}$$

Similarly, the average energy consumption of tasks is given by

$$E^{avg} = \frac{E^{total}}{N \cdot T} \tag{13}$$

where E^{total} denotes the total energy consumption, calculated as

$$E^{total} = \sum_{t=0}^{T-1} \sum_{n=1}^{N} \Big[(1 - \sum_{m \in \mathcal{M}} x_{n,m}(t)) \cdot e_n^{local}(t) + x_{n,m}(t) \cdot e_n^{rsu}(t) \Big] \tag{14}$$

In this paper, we aim to minimize the weighted sum of average task time and average energy consumption while considering the load balancing among RSUs. Thus, the objective function $J(x, y, F)$ can be defined as:

$$J(x, y, F) = \omega \cdot T^{avg} + (1 - \omega) \cdot E^{avg}, \tag{15}$$

where ω ($0 \leq \omega \leq 1$) is a weighting factor to balance the trade-off between task time and energy optimization. The optimization problem is formulated as:

$$\text{P1:} \quad \min_{x,y,F} J(x,y,F)$$

$$\text{s.t.} \sum_{m=1}^{M} x_{n,m}(t) \leq 1, \forall n \in \mathcal{N} \tag{16a}$$

$$\sum_{m'=1}^{M} y_{n,m'}(t) \leq 1, \forall n \in \mathcal{N} \tag{16b}$$

$$F_{n,m}(t) \geq 0, \forall n \in \mathcal{N}, \forall m \in \mathcal{M} \tag{16c}$$

$$\sum_{n=1}^{N} y_{n,m'}(t) \cdot F_{n,m'}(t) \leq F_{m'}, \forall m' \in \mathcal{M} \tag{16d}$$

$$LBM(t) \leq \delta + \varphi(t), \forall t \in \mathcal{T} \tag{16e}$$

$$x_{n,m}(t) \in \{0,1\} \tag{16f}$$

$$y_{n,m'}(t) \in \{0,1\} \tag{16g}$$

Constraint (16a) restricts each task to offloading to at most one RSU. Constraint (16b) ensures each task is processed on a single device. Constraint (16c) requires non-negative resource allocation for task. Constraint (16d) caps the total computational resources allocated by a server to all tasks within its capacity. Constraint (16e) keeps the standard deviation of RSU loads within an acceptable range. Constraints (16f) and (16g) define the valid ranges for the offloading and migration variables, respectively.

Notably, the optimization problem P1 aims to minimize the weighted total cost over the entire time horizon, which constitutes a joint optimization problem with a long-term objective. However, the problem exhibits an inherent characteristic that allows it to be decomposed on a per-timeslot basis. Specifically, in any given time slot t, the parameters of newly generated tasks, such as computational workload and data size, are fixed. Consequently, the offloading decision $x(t)$, migration decision $y(t)$, and resource allocation decision $F(t)$ made in the current slot primarily impact the immediate cost (i.e., latency and energy consumption) of this slot.

Although the RSU load state $\mathcal{L}_m(t)$ carries over to influence the decision-making environment of the subsequent time slot, this structure aligns perfectly with the properties of a Markov Decision Process(MDP). In an MDP, the current action determines not only the immediate reward but also the next state of the system. Based on this observation, we reformulate the original long-term optimization problem P1 as a sequential decision problem. In each time slot, the agent makes decisions based on the observed system state to minimize the immediate cost, which corresponds to the component of the total cost in Eq. (15) for that slot. By optimizing the immediate cost at each step, we can achieve the optimization of the long-term cumulative objective. This decomposition approach transforms the complex global optimization problem into a series of subproblems that can be addressed at each time step, thereby establishing the theoretical foundation for applying deep reinforcement learning to find a solution.

B. RSU Resource Allocation

Once the discrete offloading and migration decisions (i.e., the variables $x(t)$ and $y(t)$) are determined in a given time slot t, the original problem P1 reduces to a resource allocation subproblem that depends solely on the continuous variable $F(t)$. The objective of this subproblem is to find the optimal CPU resource allocation for all tasks offloaded within the current time slot.

To facilitate a clearer formulation, we introduce the set $\mathcal{N}_t = \{n \in \mathcal{N} \mid \sum_{m \in \mathcal{M}} x_{n,m}(t) = 1\}$, which represents the set of vehicles that offload their tasks to an RSU in time slot t. Consequently, for time slot t, the resource allocation subproblem P2 can be reformulated as:

$$P2: \quad \min_{F} G_t(F) = \sum_{n \in \mathcal{N}_t} \left(\omega \cdot \frac{C_n(t)}{F_{n,m'}(t)} + (1-\omega) \cdot \kappa_{m'} \cdot \left(F_{n,m'}(t)\right)^2 \cdot C_n(t) \right)$$
$$\text{s.t.} \quad (16c), (16d), (16e) \tag{17}$$

Furthermore, the objective function of P2 with respect to the resource allocation variable F for RSUs is a convex problem, convex, which can be proved as follows. The second-order partial derivative of the objective function G_t with respect to the variable F is given by

$$\frac{\partial^2 G_t}{\partial F_{n,m'}(t) \partial F_{n',m''}(t)} = \begin{cases} 2\omega \cdot C_n(t) \cdot F_{n,m'}^{-3}(t) + 2(1-\omega) \cdot \kappa_{m'} \cdot C_n(t) > 0, \\ \qquad (n,m') = (n',m'') \\ 0, \quad otherwise \end{cases} \tag{18}$$

With all off-diagonal elements being zero, the Hessian is positive semi-definite, confirming that G_t is convex with respect to $F_{n,m'}(t)$. Therefore, the optimal resource allocation F^* can be efficiently determined by leveraging standard convex optimization tools.

C. Offloading And Migration Decision Based On PPO

Having established that the resource allocation subproblem P2 is a convex problem that can be solved efficiently for any given offloading and migration decisions, the core challenge shifts to determining these discrete decision variables, $x(t)$ and $y(t)$. This decision-making process is inherently a sequential problem, as decisions made in the current time slot directly influence future system states, such as RSU loads. Furthermore, the high mobility of vehicles and the stochastic nature of task arrivals create a highly dynamic environment. Given the resulting high-dimensional state space and the combinatorial action space, traditional optimization methods are often intractable. DRL thus emerges as an ideal paradigm for solving such problems, as it excels at learning optimal policies through interaction in complex and dynamic environments.

To apply DRL to our problem, we must first formally model the sequential decision-making process as a MDP. This provides a mathematical framework for the agent to interact with the environment and learn an optimal policy. The core components of our MDP are defined as follows.

State: The state, denoted as $S = \{S_V, S_R\}$, encompasses the status of all vehicles and RSUs on the platform. Specifically, S_V encompasses each vehicle's position, road segment identifier, computational demand, data size, maximum delay tolerance, and candidate RSUs for task offloading. The candidate RSUs are defined as follows: assuming a given location can be covered by up to β RSUs, the candidate RSU sequence has a length of $\beta + 1$, with the last element representing the vehicle's local computing node. If the actual number of RSUs covering the location is less than β, the sequence is extended to length β by cyclic padding, ensuring uniformity in the length of candidate RSU sequences. Additionally, S_R incorporates the CPU computational capacity and location of each RSU.

Action: The action determines the task offloading strategy for each vehicle, specifying the initial RSU and the target RSU for task execution. The action is represented as $a = [a_{11}, a_{12}, a_{21}, a_{22}, \cdots, a_{N1}, a_{N2}]$, where a_{n1} and a_{n2} denote the initial and target RSUs for vehicle n, respectively, reflecting decisions related to task offloading and migration.

Reward: The reward serves as a performance metric for the agent, indicating the system's ability to optimize computational efficiency and resource utilization. The optimization objective can be transformed into a reward function, defined as $r = -J_t$, where J_t represents the immediate cost incurred at time step t.

With the problem formally defined as a MDP, the next step is to select and detail an algorithm capable of finding an optimal policy π_θ within this framework. For this purpose, we employ PPO, a state-of-the-art DRL algorithm operating within the Actor-Critic framework.

PPO is specifically chosen for its robust performance and enhanced training stability compared to standard Actor-Critic methods. This stability is achieved by using a novel clipped surrogate objective function, which constrains the size of policy updates at each training step, thereby preventing potentially destructive updates that can lead to performance collapse.

Our PPO implementation relies on Generalized Advantage Estimation (GAE) to compute the advantage function, which strikes an effective balance between bias and variance using a smoothing parameter λ. In practice, we collect trajectory snippets of a fixed length K from the environment and efficiently compute the GAE values over these snippets using a backward recursive method.

First, for each time step t within a trajectory snippet, we calculate its Temporal Difference (TD) error, δ_t:

$$\delta_t = r_t + \gamma V_\phi(s_{t+1})(1 - \text{done}) - V_\phi(s_t) \tag{19}$$

where γ is the discount factor, and the done flag is 1 if s_{t+1} is a terminal state and 0 otherwise.

Algorithm 1 PPO-based for task offloading and migration

Require: VECEnvironment's state, hyperparameters
Ensure: Task offloading and migration policy
Initialize actor network π_θ, critic network V_ϕ, replay buffer
Set $t \leftarrow 0$;
Initialize state $s_t \leftarrow$ env.reset();
repeat
  Get action $a_t \leftarrow \pi_\theta(s_t)$;
  Execute a_t in env:
    handle task offloading (local or RSU);
    optimize RSU CPU allocation;
    get r_t, s_{t+1}, done, info;
  Store transition $(s_t, a_t, r_t, s_{t+1}, \text{done})$;
  $t \leftarrow t + 1$;
  if replay buffer size $\geq$ MIN_BUFFER_SIZE **then**
    Sample mini-batch of MINI_BATCH_SIZE;
    for epoch = 1 **to** EPOCHS **do**
      Calculate GAE using (20);
      Compute policy ratio $r_t(\theta)$ using (21);
      Update actor network π_θ using (22);
      Update critic network V_ϕ using (23);
    end for
  end if
until $t \geq MT$
return π_θ;

Subsequently, we start from the end of the trajectory snippet and recursively compute the advantage value $\hat{A}_t$ for each time step in reverse:

$$\hat{A}_t = \sum_{j=0}^{k-1} (\gamma\lambda)^j \delta_{t+j} \tag{20}$$

This recursive relationship implies that the advantage at the current time step is the sum of the current TD error and the discounted advantage of the next time step.

During the update phase, the actor network, π_θ, is optimized by maximizing a Clipped Surrogate Objective Function. First, the probability ratio between the new and old policies is calculated:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{21}$$

Based on this, the actor's objective function, $L^{\text{CLIP}}(\theta)$, is defined as:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{22}$$

where $\hat{\mathbb{E}}_t$ denotes the expectation over a batch of samples, and ϵ is the clipping hyperparameter that constrains the magnitude of the policy update.

Concurrently, the critic network, V_ϕ, is updated by minimizing the Mean Squared Error (MSE) between its predictions and a learning target. The learning target, or the empirical return R_t, is defined as the sum of the GAE advantage and the current

state-value estimate: $R_t = \hat{A}_t + V_\phi(s_t)$. Therefore, the critic's loss function, $L_V(\phi)$, is defined as:

$$L_V(\phi) = \mathbb{E}_t \left[(V_\phi(s_t) - R_t)^2 \right] \tag{23}$$

By alternately optimizing these objective and loss functions, we combine the stability of the PPO algorithm with a problem-specific network architecture to effectively solve the complex task offloading and migration problem. Algorithm 1 summarizes the complete procedure of the PPO-based algorithm for task offloading and migration.

V. Simulations Results

In this section, simulation experiments are conducted using Python to validate the effectiveness of the proposed method. Three schemes are selected for comparative analysis:

AC: An AC agent is employed to make decisions regarding task offloading and migration for vehicles, optimizing task allocation between vehicles and RSUs.

Greedy Computation: This algorithm prioritizes task offloading based on computational demands. Tasks with higher computational requirements are preferentially allocated to RSUs for processing, while those with lower demands are processed locally by vehicles.

Greedy Data: In contrast to the computation-based approach, this algorithm prioritizes task offloading based on data size. Tasks with smaller data sizes are preferentially offloaded to RSUs to minimize transmission overhead and latency.

A. Simulation Setup

In the simulation experiment, a vehicular edge computing environment is established, comprising 100 vehicles and 11 RSUs. Vehicles move within a 1200 m × 1600 m rectangular area, with speeds uniformly distributed between 5 m/s and 10 m/s. RSUs are strategically deployed at specific coordinates to ensure effective coverage of the entire simulation area.

Regarding task characteristics, each vehicle generates computational tasks with computational demands uniformly distributed between 300 Megacycles and 450 Megacycles, and data sizes ranging from 200 KB to 300 KB. The maximum allowable delay for tasks is set between 0.5 s and 1.0 s to meet real-time requirements.

In terms of hardware configuration, vehicles are equipped with CPUs of 0.5 GHz computational capacity, while RSUs are equipped with more powerful 15 GHz CPUs. Additionally, RSUs are allocated 100 MHz of bandwidth for task processing.

B. Simulation Results

Fig. 2 illustrates the variation in reward values for the proposed method and the AC method during the training process. As shown, the proposed method exhibits a rapid increase in reward values in the early training stages, rising from -0.375 to approximately -0.265, indicating strong convergence. In contrast, the AC method demonstrates a slower improvement over the same period, with reward values increasing from -0.375 to only -0.285. These results suggest that the proposed method can more efficiently adapt to dynamic environments and rapidly optimize task offloading and migration strategies.

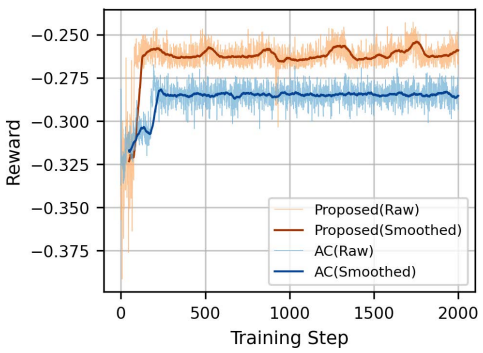

Fig. 2. Training processes comparison of with different method.

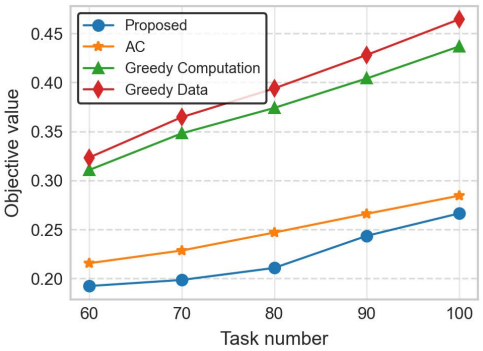

Fig. 3. Comparison of the objective value with the different method.

The overall system cost, reflected by the objective function value, was evaluated for all four methods under increasing task loads, with the results plotted in Fig. 3. The proposed method consistently outperforms others, achieving a value of 0.27 at 100 tasks, compared to 0.29 for the AC method, and 0.44 and 0.47 for the Greedy Computation and Greedy Data methods, respectively. This superiority stems from the proposed method's dynamic optimization of task offloading and migration using DRL, reducing latency and energy use.

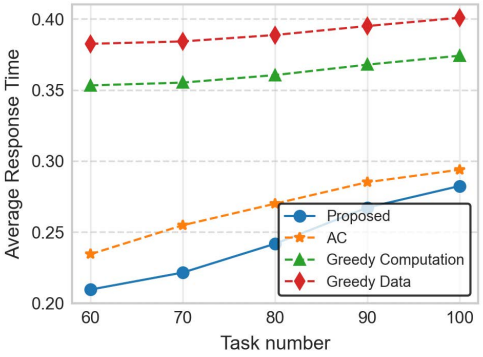

Fig. 4. Comparison of response time with different numbers of vehicles.

In contrast, the greedy methods struggle with dynamic task variations, leading to inefficient resource allocation under high loads. Combined with training analysis in Fig. 2, the proposed method's lower objective function values confirm its efficiency in dynamic settings, supporting its practical applicability.

To further assess performance from a user's perspective, the average task response time was analyzed, as detailed in Fig. 4. The proposed method achieves a response time of 0.28 seconds at 100 tasks, outperforming others due to its dynamic task offloading, prioritizing high-demand tasks to resource-sufficient RSUs. The AC method records 0.31 seconds, hindered by less effective strategy adjustments under resource contention. The Greedy Computation and Greedy Data methods yield higher response times of 0.37 and 0.41 seconds, respectively, due to single-criterion decision-making that neglects RSU load balancing, causing overload and delays. The proposed method's optimized resource allocation ensures low latency in high-load scenarios, highlighting its robustness.

VI. Conclusion

In this paper, we propose an optimization method for VEC systems based on PPO to address performance degradation due to high vehicle mobility, dynamic task arrivals, and uneven load distribution among RSUs. By integrating an adaptive mechanism for task offloading and migration with the PPO deep reinforcement learning algorithm, this study achieves joint optimization of task scheduling and resource allocation, effectively reducing system latency and energy consumption while enhancing task completion rates. Simulation results show that, compared to greedy algorithms and the AC algorithm, the PPO-based approach demonstrates superior convergence and adaptability in dynamic environments, significantly reducing average response time and energy consumption while ensuring load balancing among RSUs. This method provides an efficient and robust solution for resource scheduling and task management in the IoV and ITS, thereby advancing the development and application of related technologies.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (62271486, 62476276, and 62071327), the Emerging Frontiers Cultivation Program of Tianjin University Interdisciplinary Center, and the Tianjin Science and Technology Planning Project (22ZYYYJC00020). Chaogang Tang is the corresponding author.

References

[1] H. Min, A. M. Rahmani, P. Ghaderkourehpaz, K. Moghaddasi, and M. Hosseinzadeh, "A joint optimization of resource allocation management and multi-task offloading in high-mobility vehicular multi-access edge computing networks," *Ad Hoc Networks*, vol. 166, p. 103656, 2025.

[2] C. Tang, Z. Li, S. Xiao, H. Wu, and W. Chen, "A bandwidth-fair migration-enabled task offloading for vehicular edge computing: a deep reinforcement learning approach," *CCF Transactions on Pervasive Computing and Interaction*, vol. 6, no. 3, pp. 255–270, 2024.

[3] L. Lu, J. Yu, H. Du, and X. Li, "A3c-based load-balancing solution for computation offloading in sdn-enabled vehicular edge computing networks," *Peer-to-Peer Networking and Applications*, vol. 16, no. 2, pp. 1242–1256, 2023.

[4] D. Jiang, B. Zhu, X. Liu, and S. Mumtaz, "Alcod: An adaptive load-aware approach to load balancing for containers in iot edge computing," *IEEE Internet of Things Journal*, 2024.

[5] S. Moon and Y. Lim, "Client selection for federated learning in vehicular edge computing: A deep reinforcement learning approach," *IEEE Access*, 2024.

[6] T. Z. Gebrekidan, S. Stein, and T. J. Norman, "Combinatorial client-master multiagent deep reinforcement learning for task offloading in mobile edge computing," in *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '24. International Foundation for Autonomous Agents and Multiagent Systems, 2024, p. 2273–2275.

[7] C. Gong, L. Wei, D. Gong, T. Li, and F. Feng, "Energy-efficient task migration and path planning in uav-enabled mobile edge computing system," *Complexity*, vol. 2022, no. 1, p. 4269102, 2022.

[8] N. Agrawal, "Dynamic load balancing assisted optimized access control mechanism for edge-fog-cloud network in internet of things environment," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 21, p. e6440, 2021.

[9] B. Xie and H. Cui, "Deep reinforcement learning-based dynamical task offloading for mobile edge computing," *The Journal of Supercomputing*, vol. 81, no. 1, p. 35, 2025.

[10] W. Yang, Z. Liu, X. Liu, and Y. Ma, "Deep reinforcement learning-based low-latency task offloading for mobile-edge computing networks," *Applied Soft Computing*, vol. 166, p. 112164, 2024.

[11] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581–2593, 2019.

[12] W. Yang and Z. Liu, "Efficient vehicular edge computing: A novel approach with asynchronous federated and deep reinforcement learning for content caching in vec," *IEEE Access*, vol. 12, pp. 13 196–13 212, 2024.

[13] Y. Cang, M. Chen, J. Zhao, T. Gong, J. Zhao, and Z. Yang, "Fair computation efficiency for ofdma-based multiaccess edge computing systems," *IEEE Communications Letters*, vol. 27, no. 3, pp. 916–920, 2022.

[14] J. Wan, B. Chen, S. Wang, M. Xia, D. Li, and C. Liu, "Fog computing for energy-aware load balancing and scheduling in smart factory," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4548–4556, 2018.

[15] H. Yu and Q. Zhang, "Hybrid learning based service migration for cost minimization with deadlines in multi-user mobile edge computing systems," *Computer Networks*, vol. 242, p. 110249, 2024.

[16] Q. Shen, B.-J. Hu, and E. Xia, "Dependency-aware task offloading and service caching in vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 12, pp. 13 182–13 197, 2022.

[17] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Minimizing the Delay and Cost of Computation Offloading for Vehicular Edge Computing," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2897–2909, Sep. 2022.

[18] C. Cheng, L. Zhai, X. Zhu, Y. Jia, and Y. Li, "Dynamic task offloading and service caching based on game theory in vehicular edge computing networks," *Computer Communications*, vol. 224, pp. 29–41, Aug. 2024.

[19] S. Cao, D. Liu, C. Dai, C. Wang, Y. Yang, W. Zhang, and D. Zheng, "Reinforcement learning based tasks offloading in vehicular edge computing networks," *Computer Networks*, vol. 234, p. 109894, Oct. 2023.

[20] M. Ahmed, S. Raza, H. Ahmad, W. U. Khan, F. Xu, and K. Rabie, "Deep reinforcement learning approach for multi-hop task offloading in vehicular edge computing," *Engineering Science and Technology, an International Journal*, vol. 59, p. 101854, Nov. 2024.

[21] Z. Wu, Z. Jia, X. Pang, and S. Zhao, "Deep Reinforcement Learning-Based Task Offloading and Load Balancing for Vehicular Edge Computing," *Electronics*, vol. 13, no. 8, p. 1511, Jan. 2024.

[22] X. Zhu, W. Yao, and W. Wang, "Load-aware task migration algorithm toward adaptive load balancing in Edge Computing," *Future Generation Computer Systems*, vol. 157, pp. 303–312, Aug. 2024.

[23] Y. Liu, Q. Meng, K. Chen, and Z. Shen, "Load-aware switch migration for controller load balancing in edge–cloud architectures," *Future Generation Computer Systems*, vol. 162, p. 107489, Jan. 2025.