sensors

MDPI

Article

GAPO: A Graph Attention-Based Reinforcement Learning Algorithm for Congestion-Aware Task Offloading in Multi-Hop Vehicular Edge Computing

Hongwei Zhao, Xuyan Li *, Chengrui Li and Lu Yao

Department of Intelligent Science and Information Engineering, Shenyang University, Shenyang 110000, China
* Correspondence: lixuyan28@gmail.com

Abstract

Efficient task offloading for delay-sensitive applications, such as autonomous driving, presents a significant challenge in multi-hop Vehicular Edge Computing (VEC) networks, primarily due to high vehicle mobility, dynamic network topologies, and complex end-to-end congestion problems. To address these issues, this paper proposes a graph attention-based reinforcement learning algorithm, named GAPO. The algorithm models the dynamic VEC network as an attributed graph and utilizes a graph neural network (GNN) to learn a network state representation that captures the global topological structure and node contextual information. Building on this foundation, an attention-based Actor–Critic framework makes joint offloading decisions by intelligently selecting the optimal destination and collaboratively determining the ratios for offloading and resource allocation. A multi-objective reward function, designed to minimize task latency and to alleviate link congestion, guides the entire learning process. Comprehensive simulation experiments and ablation studies show that, compared to traditional heuristic algorithms and standard deep reinforcement learning methods, GAPO significantly reduces average task completion latency and substantially decreases backbone link congestion. In conclusion, by deeply integrating the state-aware capabilities of GNNs with the decision-making abilities of DRL, GAPO provides an efficient, adaptive, and congestion-aware solution to the resource management problems in dynamic VEC environments.

Keywords: edge computing; multi-hop networks; task offloading; V2X communication; graph neural network; deep reinforcement learning; attention

Academic Editor: Gianni D'Angelo

Received: 26 June 2025
Revised: 30 July 2025
Accepted: 4 August 2025
Published: 6 August 2025

Citation: Zhao, H.; Li, X.; Li, C.; Yao, L. GAPO: A Graph Attention-Based Reinforcement Learning Algorithm for Congestion-Aware Task Offloading in Multi-Hop Vehicular Edge Computing. *Sensors* **2025**, *25*, 4838. https://doi.org/10.3390/s25154838

1. Introduction

With the evolution of fifth-generation (5G) and future sixth-generation (6G) mobile communication technologies, emerging applications, such as autonomous driving, augmented reality, and smart cities, are transforming key sectors, such as transportation and communication [1–3]. One such application is the internet of vehicles (IoV), which demands massive data exchange and complex real-time computation between vehicles and their surroundings. However, the onboard computing units in vehicles are constrained by power consumption, cost, and physical space, making it difficult for them to independently handle these computation-intensive and delay-sensitive tasks. While traditional cloud computing solutions offer powerful computational capabilities, their centralized architecture and long communication distances result in significant transmission delays, failing to meet the stringent low-latency requirements of applications like autonomous

driving. To address this challenge, Vehicular Edge Computing (VEC) has emerged. By deploying computing and storage resources at the network edge (e.g., at roadside units, RSUs), VEC brings services closer to the end-users, thereby significantly reducing latency, alleviating core network bandwidth pressure, and becoming a key technology supporting future intelligent transportation systems.

In the VEC framework, task offloading is a core function that determines whether, where, and how to migrate a vehicle's computational tasks to edge servers for execution. However, achieving efficient task offloading in dynamic, multi-hop VEC environments faces a series of severe challenges. First, the high mobility of vehicles causes rapid changes in network topology and wireless channel conditions, making it extremely difficult to establish stable and reliable offloading links. Second, while multi-hop communication extends the range of resources available to vehicles, it also introduces more complex routing decisions and end-to-end congestion management problems. Finally, task offloading is inherently a complex multi-objective optimization problem that requires balancing multiple goals, such as minimizing task completion latency and alleviating network transmission congestion [4].

To address these challenges, researchers have proposed various technical solutions. Early work focused on traditional optimization methods and heuristic algorithms, which, despite their theoretical rigor, often struggle to adapt to the high dynamics of VEC networks. More recently, deep reinforcement learning (DRL) has become a popular approach due to its powerful decision-making capabilities. However, most existing DRL methods rely on flattened state vectors, which fail to capture the complex topological dependencies among network nodes, leading to a lack of global network awareness. While the latest research has begun to combine graph neural networks (GNNs) with DRL, showing great potential, how to systematically address the unique multi-hop link congestion problem in this paradigm remains a challenge that requires further investigation. To systematically address these challenges, this paper is motivated by a series of interconnected research questions. We investigate how to effectively model the dynamic and topologically complex VEC network to capture global state information, particularly regarding inter-node dependencies and congestion, thus forming a solid foundation for intelligent decision-making. Building on this, we explore how to design a sophisticated decision-making mechanism capable of navigating the hybrid action space, which involves not only selecting a discrete offloading destination but determining continuous ratios for task offloading and resource allocation. Furthermore, we examine how to structure a learning framework guided by a multi-objective reward function to effectively balance the conflicting goals of minimizing task latency and mitigating network congestion. Ultimately, this study seeks to validate that such an integrated approach can yield a solution that is significantly more efficient, adaptive, and robust than traditional heuristic and standard reinforcement learning methods, especially under varying network scales and loads. A detailed review and analysis of these related works will be presented in Section 2. This paper proposes an intelligent, distributed task offloading algorithm based on graph attention policy optimization (GAPO). This algorithm deeply integrates the powerful representation learning capabilities of graph neural networks with the sequential decision-making abilities of deep reinforcement learning.

The main contributions of this paper are summarized as follows:

1. We propose a novel GNN-based state encoder that effectively captures global contextual information and spatiotemporal dependencies from the complex network topology, overcoming the problem of incomplete state information in traditional RL methods.

2. We design a novel attention-based policy network within an Actor–Critic framework that intelligently selects the optimal offloading destination while collaboratively determining the continuous ratios for offloading and resource allocation.
3. We develop a comprehensive multi-objective reward function that guides the agent to simultaneously minimize task completion latency and, crucially, to alleviate network congestion on multi-hop communication links.
4. Through comprehensive simulations and ablation studies, we demonstrate that GAPO significantly outperforms traditional heuristics and standard DRL approaches in both latency and congestion mitigation.

2. Related Work

The problem of task offloading in VEC has been approached from various angles, ranging from traditional optimization techniques to modern machine learning methods. This section provides a structured overview of the existing literature, contextualizing our proposed GAPO algorithm.

2.1. Traditional and Heuristic Approaches

Early attempts to solve the VEC offloading problem often relied on formal mathematical optimization. Many works have formulated the problem as a mixed-integer nonlinear programming (MINLP) problem, as seen in the work by Dai et al. [5] and Sahni et al. [6]. These formulations are comprehensive, capturing many system variables, but are generally NP-hard, making them computationally intractable for real-time decision-making in large-scale networks.

To find more tractable solutions, researchers have explored various optimization techniques. For instance, Han et al. employed semidefinite relaxation (SDR) to find an approximate solution for a heterogeneous offloading model where communication costs are asymmetric, providing a theoretical performance guarantee under certain conditions [7]. Other approaches have utilized Lyapunov optimization to design online algorithms that can make decisions without requiring knowledge of future system states, decomposing long-term objectives into a series of per-timeslot decisions [8]. While powerful for ensuring system stability, these methods may not fully capture the complex spatial dependencies inherent in multi-hop network topologies. Heuristic algorithms, such as the greedy lowest load method (one of our baselines), offer low-complexity solutions but often lead to suboptimal, myopic decisions that can cause network bottlenecks, such as the "hotspot" effect, where traffic is funneled towards a single powerful server. Other foundational approaches in intelligent vehicle systems have relied on different paradigms. For instance, some research has combined machine learning with heuristics, such as using a deep neural network to first classify vehicle states, with a subsequent heuristic algorithm making management decisions based on this learned information [9]. Another classic approach involves knowledge-based expert systems; for example, Zielonka et al. [10] developed a real-time diagnostic system using a type-2 fuzzy control system which relies on predefined expert rules to interpret sensor data. While effective for their specific, well-defined problems, these methods highlight the limitations that motivated our work. A system based on pre-set heuristics or fuzzy rules can be difficult to scale and adapt to the highly stochastic dynamics and complex topological nature of multi-hop VEC environments, a challenge our end-to-end DRL-based approach is explicitly designed to overcome.

These traditional and heuristic methods, while providing important foundational insights, often struggle with the high dimensionality, stochastic dynamics, and complex topological nature of multi-hop VEC environments.

2.2. Deep Reinforcement Learning for Task Offloading

To overcome the limitations of static models, deep reinforcement learning (DRL) has emerged as a powerful paradigm. DRL agents can learn effective policies through direct interaction with a dynamic environment, making them well-suited for VEC scenarios. Many DRL-based solutions have explored deep deterministic policy gradient (DDPG) or deep Q-networks (DQN) for resource allocation and for offloading decisions [11,12].

However, a significant limitation of these standard DRL approaches is their reliance on a "flattened state vector" to represent the network environment. This approach struggles to capture the complex topological structure and spatiotemporal dependencies among nodes. As we demonstrated in our ablation study (Section 5.3), this loss of topological information prevents the agent from perceiving multi-hop link congestion, leading to poor performance.

2.3. Emerging GNN-Based DRL Methods

Recognizing the limitations of flattened state representations, a recent and promising research trend is the integration of graph neural networks (GNNs) with DRL. GNNs are inherently designed to operate on graph-structured data, making them a natural fit for modeling complex network topologies and learning context-aware node representations [13,14].

Several studies have begun to explore this synergy. For example, Wu et al. proposed a heterogeneous graph attention network (HGAT) combined with DRL to handle dependency-aware task offloading, where tasks are modeled as directed acyclic graphs (DAGs) [15]. Their work excels at managing complex subtask dependencies. By contrast, our proposed GAPO algorithm focuses on a different but equally critical challenge: managing inter-node resource competition and mitigating backbone link congestion in a multi-hop environment. While both approaches leverage GNNs, their problem formulations and architectural focus differ, addressing complementary aspects of the VEC offloading problem. The core idea of GAPO is to use the GNN to learn a state representation that explicitly encodes network-wide congestion, enabling the DRL agent to make globally-aware, non-greedy decisions.

To further clarify our contribution, it is crucial to differentiate our work from GNN applications in related domains, such as the traffic forecasting model proposed by Polowczyk et al. [14] While both our work and theirs leverage GNNs to capture spatial dependencies in a network—be it communication links or city streets—the fundamental problem and technical approach are distinct. Polowczyk et al. focused on a predictive task: forecasting future traffic conditions. They ingeniously combined a GNN with an LSTM within a supervised learning framework to predict future values. Their model answers the following question: "What will the network state be?"

In stark contrast, GAPO addresses a sequential decision-making and control problem: task offloading in Vehicular Edge Computing. Our framework does not predict the future state; it learns a policy to take optimal actions in the current state. In our approach, the GNN functions as a powerful state encoder for a deep reinforcement learning (DRL) agent. Its role is to create a rich, context-aware representation of the current network environment, including computational loads and link congestion. This state representation is then fed into our attention-based Actor–Critic network, which answers the following question: "Given the current state, what is the best action to take now?" Our agent learns this decision-making process through trial-and-error interaction with the environment, guided by a reward signal, which is a fundamentally different paradigm from supervised forecasting.

Therefore, while Polowczyk et al. provided a valuable tool for traffic prediction, our work introduces a complete DRL-based control framework that utilizes GNN-encoded state information to actively manage network resources and to make complex, multi-objective offloading decisions in real-time.

2.4. Comparison with State-of-the-Art and GAPO's Contributions

To further contextualize our work, we compare GAPO methodologically with two recent state-of-the-art solutions. F. Busacca et al. proposed MANTRA [16], a distributed framework using multi-armed bandits (MABs) where battery-powered RSUs (M-Boxes) autonomously manage their power states and offloading decisions. MANTRA's MAB-based approach is highly effective for the discrete action problem of selecting an offloading target. However, GAPO addresses a more complex hybrid action space; our attention-based Actor network not only selects where to offload (a discrete action) but jointly determines how much to offload (a continuous ratio) and how much resource to allocate (another continuous ratio), providing a more fine-grained, integrated decision-making process.

S. Chen et al. [17] formulated the offloading problem as a multi-objective optimization problem (MOOP) and solved it using a novel hybrid genetic algorithm. Evolutionary algorithms like this are powerful for finding a set of Pareto-optimal solutions for a given state. By contrast, GAPO is designed for sequential, real-time decision-making. Our DRL framework learns a policy that maps the current network state to an immediate, effective action, which is inherently more adaptive to the step-by-step evolution of a dynamic VEC environment.

3. System Model

This section provides a detailed exposition of the system model for a multi-hop VEC network. It covers the network architecture, graph representation, task characteristics, latency models, and the offloading decision framework. Additionally, it presents the communication and computation models used to evaluate task processing latency and resource utilization. This section provides a detailed exposition of the system model for a multi-hop VEC network. It covers the network architecture, graph representation, task characteristics, latency models, and the offloading decision framework. To enhance clarity and to provide a convenient reference, the main notations used throughout this section are summarized in Table 1.

Table 1. Summary of Key Notations.

Symbol	Description
$\mathcal{V}$	Set of all vehicles in the system.
$\mathcal{R}$	Set of all roadside units.
$\boldsymbol{\mathcal{G}_t}$	The attributed graph representing the network state at time t.
$\boldsymbol{\mathcal{N}}_t$	Set of all nodes at time t.
$\boldsymbol{\mathcal{E}}_t$	Set of all communication links at time t.
$\boldsymbol{\mathcal{X}}_t, \mathbf{E}_t$	Node and edge attribute matrices.
$\mathcal{T}_i$	The i-th computational task.
d_i	Data size of task $\mathcal{T}_i$(in KB).
c_i	Computational complexity of task $\mathcal{T}_i$ (in giga-cycles).
A_i	The joint action taken for task $\mathcal{T}_i$.
$\mathcal{K}_i$	Discrete offloading destination for task $\mathcal{T}_i$.
λ_i	Continuous offloading ratio for task $\mathcal{T}_i$.
α_i	Continuous computational resource allocation ratio for task $\mathcal{T}_i$.
$\mathcal{L}_{local}$	Latency for local computation.
$\mathcal{L}_{edge}$	Latency for computation on an edge server.
$\mathcal{L}_{v2r}, \mathcal{L}_{r2r}$	Communication latency over a V2R or R2R link.
$\mathcal{L}_i^{total}$	Total end-to-end completion latency for a task.
$\mathcal{L}_{avg}, \mathcal{C}_{cong}$	Average task latency and average R2R link congestion.
$\mathcal{J}(\mathbf{X})$	The multi-objective function to be minimized.

3.1. System Architecture

We consider a multi-hop VEC network environment designed to provide low-latency, high-reliability computation services for mobile vehicles. As shown in Figure 1, the system architecture primarily consists of three core components: vehicles, roadside units (RSUs), and the communication links between them. As depicted in Figure 1, the scenario involves heterogeneous RSU loads, where RSU1–RSU3 serve more vehicles than RSU4. This imbalance can cause vehicles to offload tasks to the less-loaded RSU4, which in turn causes the transmission links related to RSU4 to become congested. GAPO aims to achieve lower link congestion rates while ensuring low latency.

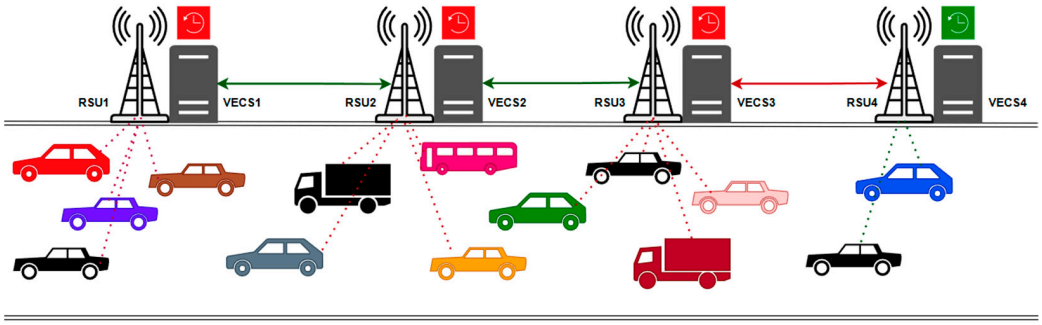

Figure 1. Multi-hop Vehicular Edge Computing Network.

We consider a one-way highway of length L_{road}. The vehicle layer is composed of a set of dynamically moving vehicles, denoted as $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$, where each vehicle $v_n \in \mathcal{V}$ is treated as a mobile computation node. Vehicles are the primary initiators of computational tasks. These tasks are typically computation-intensive and latency-sensitive, such as path planning in real-time navigation, image rendering in augmented reality applications, or environmental perception data analysis in autonomous driving. Each vehicle v_n is equipped with a limited on-Board unit (OBU) with a computational capacity denoted by $\mathcal{F}_{v_n}^{local}$, measured in giga-cycles per second. Due to constraints in size, power consumption, and cost, the local computational capability of a vehicle is far inferior to that of dedicated edge servers. Vehicles are equipped with wireless communication modules, supporting communication with other vehicles (vehicle-to-vehicle, V2V) and with roadside infrastructure (vehicle-to-infrastructure, V2I), especially RSUs. The edge computing layer consists of a set of roadside units deployed along the road, denoted as $\mathcal{R} = \{r_1, r_2, \ldots, r_M\}$, where M is the total number of RSUs. RSUs act as bridges connecting vehicles to the core network and provide edge computing resources. Compared to the vehicles, each RSU $r_m \in \mathcal{R}$ possesses significantly more powerful computing servers, with its computational capacity denoted as $\mathcal{F}_{r_m}^{RSU}$. These RSUs can efficiently process computation-intensive tasks offloaded from the vehicles.

We consider the heterogeneity of the RSU computational capacity, meaning different RSUs may have different processing speeds, configured from a predefined set based on their deployment location and expected load. In this paper, we assume RSUs provide network access to vehicles within their coverage area via the IEEE 802.11p [18] wireless interface. RSUs are interconnected through a high-speed, low-latency backbone network, forming a multi-hop collaborative network. This RSU-to-RSU (R2R) connectivity allows for the forwarding and migration of tasks or their results between different RSUs, thereby enabling broader resource sharing and more flexible load balancing. However, for such multi-hop networks, the problem of link congestion becomes highly prominent. In our model, we assume the RSUs form a linear topology based on their geographical locations, and the R2R links have a base bandwidth of $\mathcal{B}^{r2r}$.

3.2. Network Graph Representation

To effectively model and analyze the complex multi-hop VEC system, we abstract it into a dynamic, attributed graph structure. At any given time t, the network can be formally represented as a weighted graph $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{E}_t, \mathcal{X}_t, \mathbf{E}_t)$. $\mathcal{N}_t$ represents the set of nodes, which includes the set of all vehicles within the system's service area at time t, $\mathcal{V}_t = \{v_1, v_2, \ldots, v_{\mathcal{N}_t}\}$, and the set of RSUs, $\mathcal{R}$. Here, $\mathcal{N}_t$ is the number of active vehicles at that time. Due to vehicle mobility, $\mathcal{V}_t$ is a time-varying set. $\mathcal{E}_t$ represents the set of edges, which indicates the communication links existing between nodes at time t. We denote the V2R links between vehicles and RSUs as $\mathcal{E}_t^{v2r} = \{(v_n, r_m) | v_n \in \mathcal{V}_t, r_m \in \mathcal{R}\}$. The establishment and termination of these links are directly affected by vehicle locations and RSU coverage ranges, and are therefore time-varying. The R2R links between RSUs are denoted as $\mathcal{E}^{r2r} = \{(r_i, r_j) | r_i, r_j \in \mathcal{R}\}$. These links are generally static, determined by the physical connections of the infrastructure. $\mathcal{X}_t \in \mathbb{R}^{|\mathcal{N}_t| \times \mathcal{D}_N}$ represents the node attribute matrix, where $\mathcal{D}_N$ is the dimension of node features. For any node $u \in \mathcal{N}_t$ its attribute vector $\boldsymbol{x}_{u,t} \in \mathbb{R}^{\mathcal{D}_N}$ contains the following information:

1. Node type: A one-hot encoding vector indicating whether the node is a vehicle or an RSU.
2. Node computational capacity.
3. Current computational load: The instantaneous CPU utilization of the node, $\rho_{cpu}(u, t)$.
4. Available computational resources: The currently unallocated computational resources of the node.
5. Geographical location information: The coordinates of the node, which are dynamic for vehicles and fixed for RSUs.
6. Node degree: The number of connections the node has in the graph.

These attributes provide rich input for the subsequent GNN, enabling it to learn an effective representation of the node's state.

$\mathbf{E}_t \in \mathbb{R}^{|\mathcal{E}_t| \times \mathcal{D}_E}$ represents the edge attribute tensor, where $\mathcal{D}_E$ is the dimension of edge features. For any edge $e = (u, w) \in \mathcal{E}_t$, its attribute vector $\boldsymbol{e}_{u,w} \in \mathbb{R}^{\mathcal{D}_E}$ includes the following:

1. Link type: A one-hot encoding indicating whether it is a V2R or R2R link.
2. Link bandwidth capacity.
3. Current link load: The instantaneous utilization of the link, $\rho_{link}(e, t)$.
4. Communication latency: The estimated transmission latency of the link.
5. Link reliability: The packet loss rate of the link.

The key characteristic of this graph representation is its dual dynamicity. Vehicle mobility causes the V2R link topology $\mathcal{E}_t^{v2r}$ and the vehicle node set $\mathcal{V}_t$ to change continuously. Concurrently, the random arrival and completion of tasks lead to real-time fluctuations in the load attributes of nodes and links. This dual dynamicity poses significant challenges for task offloading decisions. First, the complexity of the state space, with a constantly changing network state, makes traditional optimization methods difficult to apply. Second, the demand for real-time decision-making requires rapid responses to network changes to ensure service quality. Finally, local decisions need to consider the state of the global network.

3.3. Task Processing and Communication Latency Model

In the system, each vehicle $v_n \in \mathcal{V}$ dynamically generates a series of computational tasks. Each task $\mathcal{T}_i \triangleq (d_i, c_i, a_i)$ is treated as an independent computational unit, where d_i is the data size of the task, c_i is its computational complexity (measured in the number

of CPU cycles required to complete the task), and a_i is its arrival time. In this paper, task arrivals are modeled as a Poisson process.

For each task $\mathcal{T}_i$, the generating vehicle v_n must make the following decisions:

1. Offloading Destination Selection:

$$\mathcal{K}_i = \begin{cases} v_n, & \textit{if the task is processed locally on the vehicle} \\ r_m, & \textit{if the task is offloaded to } r_m \in R \end{cases} \tag{1}$$

2. Offloading Ratio:

If a task is offloaded, the agent can decide the ratio $\lambda_i \in [0, 1]$ of the task's computational workload to be offloaded.

The offloaded computation workload is as follows:

$$\mathcal{C}_i^{offload} = \lambda_i \cdot c_i \tag{2}$$

The locally processed workload is as follows:

$$\mathcal{C}_i^{local} = (1 - \lambda_i) \cdot c_i \tag{3}$$

The amount of data transmitted to the RSU is as follows:

$$\mathcal{D}_i^{tx} = \lambda_i \cdot d_i \tag{4}$$

If $\lambda_i = 0$, it represents local execution; if $\lambda_i = 1$, it represents full offloading.

3. Computational Resource Allocation Ratio:

If a portion of the task ($\lambda_i > 0$) is offloaded to RSU r_m, the agent also determines the proportion $\alpha_i \in [0, 1]$ of r_m's total computational capacity $\mathcal{F}_{r_m}^{RSU}$ that will be dedicated to processing $\mathcal{C}_i^{offload}$. This can be viewed as a form of resource reservation or priority scheduling based on slicing.

The allocated computational resource is as follows:

$$\mathcal{F}_{r_m,i}^{alloc} = \alpha_i \cdot \mathcal{F}_{r_m}^{RSU} \tag{5}$$

If the task is processed locally ($\mathcal{K}_i = v_n$), we let $\lambda_i = 0$ and $\alpha_i = 1$ for notational convenience.

Therefore, the action A_i that the agent selects for each task $\mathcal{T}_i$ is represented by a tuple $A_i \triangleq (\mathcal{K}_i, \lambda_i, \alpha_i)$, where $\mathcal{K}_i$ is the discrete offloading destination, and λ_i and α_i are continuous ratios for offloading and resource allocation, respectively.

Communication latency depends on the link type and its current load. In this study, we model the dynamic queuing behavior of V2R and R2R links. While the system conceptually aligns with a G/G/1 queue due to its general arrival and service patterns, we adopt specific, standard models for simulation and latency calculation to ensure reproducibility. Specifically, the arrival process is modeled as a Poisson process. This is implemented in our simulation by generating the inter-arrival times between consecutive tasks from an exponential distribution. For the service time and latency estimation, we use the well-known and computationally efficient M/M/1 queuing formula as a practical approximation. In our simulation, a task's data size and computational requirement are drawn from uniform distributions. The base service time is then calculated deterministically from these values. We use this service time within the M/M/1 latency formula to estimate the total time in the system, which includes queuing delay. This approach effectively captures the essential non-linear relationship where latency increases sharply with utilization, a critical feature

for modeling network congestion, while remaining tractable for a dynamic simulation environment. The use of the M/M/1 model is a deliberate methodological choice, justified on the following grounds: The M/M/1 model provides a tractable, closed-form expression for calculating latency. This is computationally essential within our deep reinforcement learning framework, where the agent must rapidly evaluate the consequences of numerous potential actions during the training process. A full-scale discrete-event simulation of a G/G/1 queue at each step would render the training of our complex agent computationally infeasible. While an approximation, the M/M/1 model effectively captures the core non-linear relationship between resource utilization and delay. It correctly reflects the critical phenomenon that latency increases exponentially as a link or the server approaches its full capacity. This is the primary system dynamic that our learning agent must understand and manage to avoid congestion. While an approximation, the M/M/1 model effectively captures the core non-linear relationship between resource utilization and delay. It correctly reflects the critical phenomenon that latency increases exponentially as a link or server approaches its full capacity. This is the primary system dynamic that our learning agent must understand and manage to avoid congestion. For V2R links, we treat each as a G/G/1 queue. Each RSU $\mathscr{r}_m \in \mathscr{R}$ has a total V2R wireless interface bandwidth, denoted as $\mathscr{B}_{\mathscr{r}_m}^{v2r}$. This bandwidth is limited and must be dynamically shared among all vehicles currently connected to and transmitting data to that RSU. From a macroscopic perspective, this means the effective data transmission rate for a single vehicle decreases as the number of competing vehicles increases.

For a specific vehicle $\mathscr{v}_n$ transmitting data to RSU $\mathscr{r}_m$, its instantaneous data rate $\mathscr{R}_{\mathscr{v}_n,\mathscr{r}_m}^{v2r}$ depends not only on the RSU's total bandwidth $\mathscr{B}_{\mathscr{r}_m}^{v2r}$ but on the channel conditions and the number of vehicles currently sharing the bandwidth. We consider a total link service capacity and treat all data flows arriving at this link as entering a shared queue. When vehicle $\mathscr{v}_n$ needs to transmit data of size $\mathscr{D}_i$ to RSU $\mathscr{r}_m$, the transmission latency is mainly composed of transmission time and queuing time.

1. Transmission Time:

$$\mathscr{L}_{tx} = \frac{\mathscr{D}_i}{\mathscr{R}_{eff}} \tag{6}$$

where $\mathscr{R}_{eff}$ is the effective data rate obtained for this transmission.

2. Queuing Time:

In this study, to obtain a computable latency estimate, we use the classic M/M/1 queuing formula as an approximation for the G/G/1 model's performance [19]. This is a common practice in engineering that preserves the core idea of queuing theory (i.e., latency increases non-linearly with utilization) while providing an easy-to-compute closed-form solution. The V2R interface of RSU $\mathscr{r}_m$ is viewed as a single server with a total service capacity of $\mathscr{B}_{\mathscr{r}_m}^{v2r}$ (bps). All vehicles wishing to upload data via this RSU form an arrival flow. Let $\mathscr{Q}_{arrival}^{v2r,m}$ be the current total data arrival rate for the V2R uplink at RSU $\mathscr{r}_m$, which is estimated using an exponentially weighted moving average (EWMA) of recent task arrival contributions. If $\mathscr{Q}_{arrival}^{v2r,m}$ is greater than or equal to the effective service rate, the link is considered highly congested, and latency will increase sharply. The total V2R bandwidth $\mathscr{B}_{\mathscr{r}_m}^{v2r}$ of the RSU is considered the service rate of this queue.

Therefore, the V2R communication latency for transmitting data of size $\mathscr{D}_i$, $\mathscr{L}_{v2r}(\mathscr{D}_i, \mathscr{r}_m)$, is calculated as follows:

$$\mathscr{L}_{v2r}(\mathscr{D}_i, \mathscr{r}_m) = \frac{\mathscr{T}_s^{v2r}}{1 - \rho_{link}^{v2r,m} + \epsilon} \tag{7}$$

where $\mathcal{T}_s^{v2r} = \frac{\mathcal{D}_i}{\mathcal{B}_{r_m}^{v2r}+\epsilon}$ is the basic service time without considering queuing, $\rho_{link}^{v2r,m}$ is the actual utilization of this V2R link, a factor less than 1 used to ensure queue stability and to reflect that real systems cannot achieve 100% utilization, and ϵ is a very small value to prevent division by zero.

To provide the agent with more fine-grained information that reflects instantaneous channel competition, we use a different calculation for the RSU's V2R link utilization $\rho_{link}^{v2r,m}$ when constructing the node features for the GNN. Specifically, when generating the node attribute vector $\boldsymbol{\mathcal{X}}_t$, this utilization is calculated as the ratio of the total arrival rate to an effective service rate adjusted by the number of competing vehicles. This allows the GNN to perceive the drop in effective bandwidth due to an increase in connected vehicles, thereby making more forward-looking offloading decisions.

R2R links typically use more stable and higher-bandwidth technologies than V2R links, such as fiber optic connections or dedicated point-to-point high-frequency wireless links. This means R2R links usually have higher data transmission rates and lower intrinsic latency. For a specific R2R link connecting RSU r_a and r_b, its base bandwidth capacity is denoted as $\mathcal{B}_{r_a,r_b}^{r2r}$. Unlike V2R links, the traffic on R2R links is often bidirectional and may carry data for multiple different tasks simultaneously. When data of size $\mathcal{D}_i$ needs to be transmitted between RSU r_a and r_b via a direct R2R link, its communication latency $\mathcal{L}_{r2r}(\mathcal{D}_i, r_a, r_b)$ is modeled similarly to V2R, also as a G/G/1 queue, with latency calculated approximately as follows:

$$\mathcal{L}_{r2r}(\mathcal{D}_i, r_a, r_b) = \frac{\mathcal{T}_s^{r2r}}{1 - \rho_{link}^{r_a,r_b} + \epsilon} \tag{8}$$

where $\mathcal{T}_s^{v2r} = \frac{\mathcal{D}_i}{\mathcal{B}_{r_a,r_b}^{r2r}+\epsilon}$ is the basic service time, and $\rho_{link}^{r_a,r_b}$ is the utilization of this R2R link.

If data needs to be transmitted through a series of RSUs via multi-hop, for example, along the path $\mathcal{P}_{R2R} = (r_s, r_1, r_2 \ldots, r_k, r_d)$, where r_s is the source RSU and r_d is the destination RSU, the total multi-hop R2R communication latency is the sum of the latencies of each link segment along the path, as follows:

$$\mathcal{L}_{multi-hop}(\mathcal{D}_i, \mathcal{P}_{R2R}) = \sum_{(r_u,r_v)\in\mathcal{P}_{R2R}} \mathcal{L}_{r2r}(\mathcal{D}_i, r_u, r_v) \tag{9}$$

To accurately reflect the dynamic load conditions of the network links, the data arrival rates on V2R and R2R links need to be continuously updated. The system employs an exponentially weighted moving average (EWMA) mechanism. When a data block of size d is assigned to a link, it contributes to that link's average arrival rate over a time window. As time passes, the contribution of old arrivals decays exponentially. After each time step Δt, the current arrival rate is multiplied by a decay factor. This dynamic update mechanism allows the model to capture changes in network congestion over time, providing more accurate link status information for the agent's decisions.

In our G/G/1 queuing model, when the arrival rate β of a communication link approaches or exceeds its effective service rate μ, the queuing latency of that link increases sharply, and the system becomes unstable. We define this state as link congestion. Specifically, we introduce a stability factor $\varphi < 1$. When the link utilization $\rho = \beta/\mu$ satisfies $\rho \geq \varphi$, we determine that the link is congested. Specifically, we introduce a stability factor $\varphi < 1$, set to 0.99 in our simulations (as shown in Table 2). This factor represents a crucial engineering principle: real-world queuing systems become unstable and experience exponential latency growth before reaching 100% theoretical utilization. By defining congestion as occurring when utilization ρ exceeds this factor, we provide a practical and safe margin to prevent system collapse and to guide the agent to learn proactively to avoid such states.

The severity of congestion can be measured by the proportion by which its actual utilization exceeds this stability threshold. Alleviating link congestion is a core optimization objective of this research. To determine the end-to-end communication latency from a vehicle to any potential target RSU, the optimal path that minimizes total latency must be found. This involves selecting an initial RSU to act as a wireless access point and then finding the lowest-latency multi-hop path through the RSU backbone network to the final destination. The procedure for computing this optimal path latency is formalized in Algorithm 1.

Table 2. Simulation Environment Parameters.

Parameter	Description	Value
Highway Length	Total length of the highway in the simulation	1000 m
RSU Coverage Radius	Signal coverage range of each RSU	200 m
Default RSU Count	Number of RSUs in the baseline scenario	5
RSU Default Compute Capacity	Set of RSU compute capacities (heterogeneous)	{15, 20, 25, 30, 35} Gcps
Vehicle Local Compute Capacity	On-board compute capacity of a single vehicle	0.25 Gcps
RSU–R2R Link Base Bandwidth	Backbone link bandwidth between RSUs	30 Mbps
Default Task Arrival Interval	Average time interval between task generations	0.1 s
Task Data Size	Range of data size to be transmitted for tasks	(512,1024) KB
Task Computation Size	Range of computation cycles required for tasks	(0.5,1.5) Gc
G/G/1 Stability Factor	Factor used to calculate the effective service rate	0.99
State Averaging Window	Time window for EWMA calculation	1 s

3.4. Computation Model and Total Latency Analysis

When vehicle $v_n \in \mathcal{V}$ decides to process task $\mathcal{C}_i^{local}$ locally, its computation latency primarily depends on the vehicle's own computational capacity and its current computation queue state. Each vehicle v_n has a fixed local computational capacity, denoted as $\mathcal{F}_{v_n}^{local}$. The vehicle's local processing unit can be viewed as a server, and all task segments needing local processing form a computational task arrival flow. We similarly model this as a G/G/1 queue to approximately estimate the local computation latency. The local computation latency for a workload of $\mathcal{L}_{local}\left(\mathcal{C}_i^{local}, v_n\right)$, is calculated as follows:

$$\mathcal{L}_{local}\left(\mathcal{C}_i^{local}, v_n\right) = \frac{\mathcal{T}_s^{local}}{1 - \rho_{cpu}^{v_n} + \epsilon} \tag{10}$$

where $\mathcal{T}_s^{local} = \frac{\mathcal{C}_i^{local}}{\mathcal{F}_{v_n}^{local} + \epsilon}$ is the basic computation service time without queuing, $\rho_{cpu}^{v_n}$ is the actual utilization of vehicle v_n's computation unit, and ϵ has the same meaning as in the communication model.

When task $\mathcal{C}_i^{offload}$ is offloaded to RSU r_m for processing, the computation latency depends on the RSU's capacity, its overall current computational load, and the computational resources allocated to this specific task. In our model, the agent also outputs a computational resource allocation ratio $\alpha_i \in [0, 1]$. Therefore, for the task portion using dedicated computational resources, its computation latency on RSU r_m, $\mathcal{L}_{edge}\left(\mathcal{C}_i^{offload}, r_m, \alpha_i\right)$, can be directly calculated as follows:

$$\mathcal{L}_{edge}\left(\mathcal{C}_i^{offload}, r_m, \alpha_i\right) = \frac{\mathcal{C}_i^{offload}}{\mathcal{F}_{r_m,i}^{alloc} + \epsilon} = \frac{\mathcal{C}_i^{offload}}{\alpha_i \cdot \mathcal{F}_{r_m}^{RSU} + \epsilon} \tag{11}$$

Algorithm 1: Multi-Hop Path Latency Calculation

Input: source vehicle v_n, target RSU r_m, task data size $\mathcal{D}_i$
Output: best path $\mathcal{P}^*$, minimum latency $\mathcal{L}^*$

Initialize $\mathcal{L}^* \leftarrow \infty$, $\mathcal{P}^* \leftarrow$ None
$R_{access} \leftarrow$ Get all RSUs directly connected to vehicle v_n
Create RSU − only subgraph $\mathcal{G}_r$
for each r_{access} in R_{access} **do**
    $\mathcal{L}_{v2r} \leftarrow$ Get V2R Latency $(v_n, r_{access}, \mathcal{D}_i)$
    if r_{access} is r_m **then**
        $\mathcal{L}_{path} \leftarrow \mathcal{L}_{v2r}$
        $\mathcal{P}_c \leftarrow [v_n, r_m]$
    else
        Define edge weights in $\mathcal{G}_r$ as R2R latency:
        for each edge (r_u, r_v) in $\mathcal{G}_r$ **do**
            $W(r_u, r_v) \leftarrow$ Get R2R Latency $(\mathcal{D}_i, r_u, r_v)$ for all edges in $\mathcal{G}_r$
        end for
        $\mathcal{P}_{v2r} \leftarrow$ Shortest Path ($\mathcal{G}_r$, source $= r_{access}$, target $= r_m$, weights $= W$)
        if $\mathcal{P}_{v2r}$ exists **then**
            $\mathcal{L}_{r2r} \leftarrow$ Calculate total latency of $\mathcal{P}_{v2r}$ using weights W
            $\mathcal{L}_{path} \leftarrow \mathcal{L}_{v2r} + \mathcal{L}_{r2r}$
            $\mathcal{P}_c \leftarrow [v_n] \oplus \mathcal{P}_{r2r}$ // $\oplus$ is path concatenation
        else
            continue
        end if
    end if
    if $\mathcal{L}_{path} < \mathcal{L}^*$ **then**
        $\mathcal{L}^* \leftarrow \mathcal{L}_{path}$
        $\mathcal{P}^* \leftarrow \mathcal{P}_c$
    end if
end for
return $\mathcal{P}^*$, $\mathcal{L}^*$

Although this part of the computation uses specifically allocated resources, it still consumes the RSU's actual computational capacity. Therefore, when calculating the overall arrival rate of computation for RSU r_m, this offloaded workload $\mathcal{C}_i^{offload}$ will still contribute to the RSU's average computational load. This means that even if a task obtains a dedicated resource slice, it still affects the background load perceived by subsequent tasks on that RSU, which will be reflected in the node features observed by the GNN.

Total task latency is a key metric for evaluating system performance. For a given task $\mathcal{T}_i \triangleq (d_i, c_i, a_i)$, its total processing latency $\mathcal{L}_i^{total}$ depends on the offloading decision $(\mathcal{K}_i, \lambda_i, \alpha_i)$ made by the agent. The arrival time of task $\mathcal{T}_i$ is a_i, and its completion time is $a_i + \mathcal{L}_i^{total}$.

Based on the offloading decision model defined above, we analyze the total task latency in different scenarios:

1. Fully Local Execution:

When the task is processed entirely on the source vehicle v_n, the input data d_i does not need to be transmitted, and the computation workload is c_i. The total latency is the local computation latency, which is calculated as follows:

$$\mathcal{L}_i^{total} = \mathcal{L}_{local}\left(\mathcal{C}_i^{local}, v_n\right) \tag{12}$$

2. Full Offloading to an RSU:

When the task is fully offloaded to RSU r_m, the vehicle must first transmit the input data of size d_i via V2R and potentially R2R multi-hop links to RSU r_m. Subsequently, RSU r_m uses the allocated computational resources $\mathcal{F}_{r_m,i}^{alloc} = \alpha_i \cdot \mathcal{F}_{r_m}^{RSU}$ to process the entire computation workload c_i. The total latency includes V2R communication latency, R2R multi-hop communication latency (if any), and RSU computation latency as follows:

$$\mathcal{L}_i^{total} = \mathcal{L}_{v2r}(d_i, r_m) + \mathcal{L}_{multi-hop}(d_i, \mathcal{P}_{R2R}) + \mathcal{L}_{edge}(c_i, r_m, \alpha_i) \tag{13}$$

3. Partial Offloading:

When the task is partially offloaded, its computational workload is split into a local part $\mathcal{C}_i^{local}$ and an offloaded part $\mathcal{C}_i^{offload}$. The data $\mathcal{D}_i$ needs to be transmitted to the RSU. The local computation and the offloaded computation can be performed in parallel. Therefore, the total task latency is determined by the longer of the two paths as follows:

$$\begin{aligned}\mathcal{L}_i^{total} = max\Bigg(&\mathcal{L}_{local}\left(\mathcal{C}_i^{local}, v_n\right), \mathcal{L}_{v2r}(\mathcal{D}_i, r_m) + \mathcal{L}_{multi-hop}(\mathcal{D}_i, \mathcal{P}_{R2R}) \\ &+ \mathcal{L}_{edge}\left(\mathcal{C}_i^{offload}, r_m, \alpha_i\right)\Bigg)\end{aligned} \tag{14}$$

Based on the preceding system architecture, task model, communication model, and computation model, the core problem of our research can be formulated as jointly optimizing the task offloading decisions and resource allocation strategies for a series of incoming tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$ in a dynamic multi-hop VEC network, in order to minimize a comprehensive performance metric. This metric typically includes average task processing latency, load balancing among RSUs, and network congestion levels.

Assuming the system needs to process K tasks over a time horizon, for each task $\mathcal{T}_i (i = 1, 2, \ldots, K)$, the decision variables are $\mathbf{X} = \{(\mathcal{K}_i, \lambda_i, \alpha_i)\}_{i=1}^{K}$. Our goal is to find the optimal set of decisions $\mathbf{X}^*$ that minimizes a weighted multi-objective function.

The primary optimization objectives are as follows:

1. Minimize Average Task Processing Latency, $\mathcal{L}_{avg}$:

$$\min_{\mathbf{X}} \mathcal{L}_{avg} = \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}_i^{total} \tag{15}$$

2 Minimize Average R2R Link Congestion, $\mathcal{C}_{cong}$:

This objective aims to penalize offloading decisions that cause excessive utilization of communication links. As defined in Section 3.3, congestion occurs when a link's utilization ρ_{link} exceeds the stability factor φ. We define the congestion level $\mathcal{C}_{cong}$ as the sum of the average excess utilization over all links involved in task transmissions. For a set of paths $\mathcal{P}_\pi$ generated by a policy π, this objective can be expressed as follows:

$$\min_{\mathbf{X}} \mathcal{C}_{cong} = \frac{1}{|\mathcal{P}_\pi|} \sum_{e \in \mathcal{P}_\pi} \overline{\rho_{link}^e} - \varphi \tag{16}$$

where $\overline{\rho_{link}^{e}}$ is the average utilization of link e during the evaluation period. This objective encourages traffic dispersion, avoiding over-reliance on a few R2R links, and thus alleviating potential backbone network bottlenecks.

By introducing weight factors w_1, $w_2 \geq 0$ (set to 1.0 and 0.5, respectively, in our experiments to prioritize latency while penalizing congestion), the above multiple objectives can be combined into a single scalar optimization objective function $\mathcal{J}(\boldsymbol{X})$. These weights allow a network operator to balance the critical trade-off between performance and stability. A higher w_1 prioritizes minimizing task completion time, whereas a higher w_2 enforces a more conservative policy that prioritizes avoiding network congestion, even at the cost of slightly higher latency for some tasks.

$$\min_{\mathbf{X}} \mathcal{J}(\mathbf{X}) = w_1 \cdot \mathcal{L}_{avg} + w_2 \cdot \mathcal{C}_{cong} \tag{17}$$

$$\text{s.t } \mathcal{K}_i \in [v_i] \cup \mathcal{R} \forall i \in \mathcal{N}$$

$$\lambda_i \in [0,1] \forall i \in \mathcal{N}$$

$$\alpha_i \in [0,1] \forall i \in \mathcal{N}$$

$$\rho_{cpu}^{r_m}, \rho_{link}^{e} \in [0,100]$$

$$\beta_e \leq \varphi \cdot \mu_e \forall e \in E$$

$$\mathbf{X} = \{(\mathcal{K}_i, \lambda_i, \alpha_i)\}_{i=1}^{K}$$

Formulating this problem as an optimization problem highlights its inherent complexity and challenges. The problem is NP-hard. Specifically, the formulation requires making discrete offloading decisions while optimizing the objectives and satisfying the constraints that are non-linear functions of the decision variables. This combination of discrete variables and non-linear, non-convex relationships places the problem in the class of non-convex MINLP, which is proven to be NP-hard [5]. The network state is dynamically changing over time, and may involve uncertainty, which makes traditional offline optimization methods inapplicable. Each decision affects future network states, exhibiting the characteristics of a Markov decision process (MDP). As the number of vehicles and RSUs increases, the state and action spaces grow dramatically.

4. Graph Attention Policy Optimization (GAPO)

This chapter provides a detailed introduction to our proposed GAPO algorithm, designed to solve the multi-objective task offloading and resource allocation problem in dynamic multi-hop VEC networks. We will first provide an overview of GAPO's overall framework, then elaborate on its key components, including the graph representation of the network state and GNN embeddings, the architecture of the attention-based Actor–Critic policy network, and the design of the policy optimization module.

4.1. Overview of the GAPO Algorithm Framework

As described in Section 3, multi-hop VEC environments are characterized by high dynamics and complexity: the rapid movement of vehicles leads to frequent changes in wireless channel conditions and network topology; RSU resources are limited and may be heterogeneous; and task offloading decisions must consider not only the latency of individual tasks but the overall system's load balancing and network congestion, with these objectives often being in conflict.

Based on these considerations, this paper proposes an algorithm based on graph attention policy optimization (GAPO), which aims to solve the dynamic, multi-objective,

and joint optimization problem of task offloading and resource allocation in multi-hop VEC networks. The core idea of the GAPO algorithm is to deeply integrate the powerful graph representation capabilities of graph neural networks (GNNs) with the intelligent decision-making abilities of deep reinforcement learning (DRL), and to guide the agent to learn an effective balance between multiple performance objectives through a well-designed reward mechanism. The overall framework of the GAPO algorithm is shown in Figure 2.

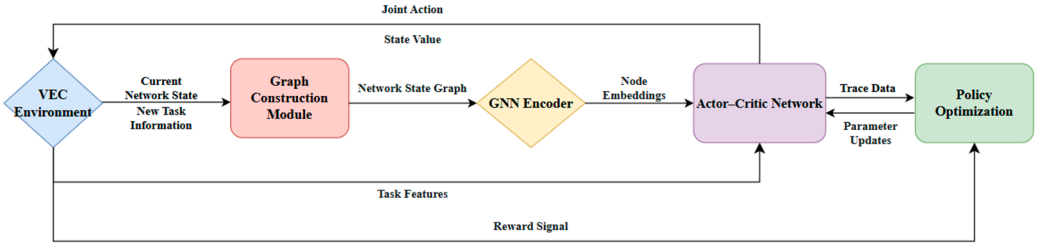

Figure 2. The GAPO Algorithm Framework.

At each decision-making moment, when a new computational task is generated, the VEC environment provides the agent with the current network state and new task information. The network state includes the real-time attributes of all vehicles and RSUs, as well as their connection relationships and link quality. The new task information describes the characteristics of the task to be processed. The received raw network state and task information are fed into a Graph Construction Module. This module is responsible for integrating this scattered information into a structured network state graph. In this graph, vehicles and RSUs are abstracted as nodes, and the communication links between them are abstracted as edges. Each node and edge is endowed with a feature vector extracted from the raw state, which may have been normalized.

The constructed network state graph is then input into a GNN encoder. The GNN encoder processes the graph data through its internal graph convolutional layers. By propagating and aggregating neighbor information on the graph, the GNN can generate a low-dimensional, dense node embedding vector for each node in the graph. These embedding vectors not only capture the intrinsic attributes of the nodes but incorporate contextual information from their network topology, thereby providing a state representation with a more global perspective for subsequent decision-making. Concurrently, the GNN encoder also outputs a state value for the network state graph, which is a preliminary assessment of the quality of the current state and can be used for subsequent advantage calculation.

The node embeddings output by the GNN encoder, along with the current task features obtained directly from the VEC environment, are fed into the Actor–Critic network. The Actor–Critic network contains two core components: the Actor and the Critic. The Actor network generates a joint action based on the received state representation. This joint action includes the discrete selection of an offloading destination and the continuous decision of resource allocation. The generated joint action is then applied back to the VEC environment, which executes the task offloading and processing according to this action and transitions to the next state. After executing the action, the VEC environment returns a reward signal to the agent. This reward signal is calculated based on a predefined multi-objective reward function, comprehensively considering performance indicators, such as task processing latency, RSU load balancing, and network congestion.

Simultaneously, the entire interaction process generates a piece of experience data, known as a trajectory, which is stored in the experience replay buffer for policy optimization. The agent periodically samples a batch of experience data from the replay buffer. Using this data, it updates the parameters of the Actor–Critic networks. The Critic network learns

to more accurately estimate the state value, while the Actor network learns to optimize its policy to maximize the cumulative expected reward.

Through the iterative steps described above, the GAPO algorithm can learn end-to-end. The GNN encoder learns how to extract effective feature representations from complex network states, while the Actor–Critic network learns how to make intelligent, multi-objective optimized decisions for task offloading and resource allocation in a highly dynamic and variable multi-hop VEC environment based on these representations.

4.2. GNN Network and Embeddings

As detailed in Section 3.2, we introduced the network graph representation. The attributed graph $\mathcal{G}_t$ constructed in this manner dynamically reflects the real-time state of the VEC network and serves as the input to the GNN encoder.

As shown in Figure 3, the core task of the GNN encoder is to learn a mapping function $\mathbb{F}_{GNN} : \mathcal{G}_t \rightarrow \mathcal{H}_t$, which transforms the input network state graph $\mathcal{G}_t$ into a set of node embeddings, represented by the matrix $\mathcal{H}_t \in \mathbb{R}^{|\mathcal{N}_t| \times \mathcal{D}_{emb}}$ is the dimension of the node embeddings. The embedding vector for each node, $h_u \in \mathbb{R}^{\mathcal{D}_{emb}}$, should encode the properties of the node itself and its contextual information within the network.

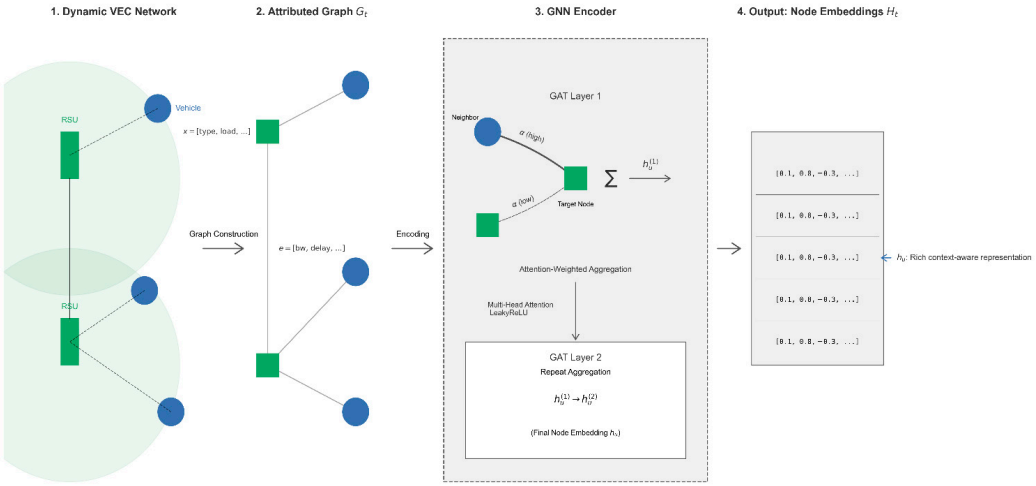

Figure 3. GNN Network and Embeddings.

In our GNN encoder, we employ a more powerful variant of the graph attention network, known as GATv2 [20], to enhance the model's expressive power. Unlike the original GAT, which utilizes a static attention mechanism, GATv2 introduces a dynamic attention mechanism that allows the model to learn more complex and context-dependent relationships between nodes. This allows it to selectively aggregate the most important neighbor information for the current task. Even if the neighbor types and feature distributions of a node vary, GATv2 can adaptively learn how to aggregate their information. Moreover, GATv2's operations are local, making it suitable for dynamically changing graph structures.

Our GNN encoder is composed of two stacked GATv2 convolutional layers. The first GATv2 layer employs multiple independent attention heads in parallel and concatenates their outputs to capture neighborhood relationships from different aspects, enhancing the model's expressive power and stability. A LeakyReLU activation function is used to introduce non-linearity. Batch normalization is applied to the concatenated features after the first GATv2 layer and before the activation function to accelerate convergence and to improve generalization. It operates on the feature dimension. Additionally, dropout is applied to the attention weights and node features to prevent overfitting. Self-loops are used to ensure that a node's own information is considered during aggregation. The second GATv2 layer typically sets the number of attention heads to 1 and directly outputs

the final embedding dimension, as this layer is directly followed by the MLP layers of the Actor–Critic network, thus no activation function is designed for it.

In each GATv2 layer l, the representation of a node u is computed using a multi-head attention mechanism. For a single attention head k, the process is as follows:

First, an attention score $e_{uv}^{(k)}$ is calculated for each neighbor v in the neighborhood of u, as follows:

$$e_{uv}^{(k)} = \left(a^{(k)}\right)^T \cdot LeakyReLU\left(W^{(k)} \cdot \left[h_u^{(\ell-1)} \middle\| h_v^{(\ell-1)}\right]\right) \tag{18}$$

where $h_u^{(\ell-1)}$ is the feature vector of node u from the previous layer, $W^{(k)}$ and $a^{(k)}$ are the learnable weight matrix and attention vector for head k, respectively, and $\|$ denotes concatenation. These scores are then normalized across all neighbors using the Softmax function to obtain the attention weights $a_{uv}^{(k)}$, as follows:

$$a_{uv}^{(k)} = \frac{exp\left(e_{uv}^{(k)}\right)}{\sum_{j \in \mathcal{N}(u) \cup \{u\}} exp\left(e_{uj}^{(k)}\right)} \tag{19}$$

Finally, the attention weights are used to aggregate the transformed features of the neighbors, producing the output for head k. For intermediate layers with k heads, we concatenate their outputs as follows:

$$h_u^{(\ell)} = \|_{k=1}^{k} \sigma\left(\sum_{v \in \mathcal{N}(u) \cup \{u\}} a_{uv}^{(k)} \cdot \left(W^{(k)} h_v^{(\ell-1)}\right)\right) \tag{20}$$

where σ is a non-linear activation function. This dynamic attention mechanism is crucial for capturing the complex dependencies in the VEC network.

After passing through two GATv2 layers of propagation and aggregation, the final output node embedding $h_u = h_u^{(2)}$ captures the structural and feature information within a two-hop neighborhood centered on node u. These node embeddings are then fed into the Actor–Critic network as the basis for their subsequent decision-making and evaluation.

The core advantage of GATv2 lies not only in its powerful representation learning capability but in its inherent interpretability. In our model, each GATv2 layer performs an attention-weighted aggregation of neighbor information. This means that, when a node updates its own feature embedding, it does not treat all neighbor nodes and its own attributes equally. Specifically, each node has key features, such as current computational load and V2R link congestion. Through end-to-end training, GATv2 learns an attention function that automatically assigns higher attention weights to neighbor nodes with more favorable resources. For example, when an RSU aggregates information, it will pay more attention to its own low computational load features and the uncongested R2R links connected to it. After several layers of GATv2 information propagation, the final output node embedding is no longer an isolated feature vector but a highly condensed, context-aware representation of its own state and the availability of surrounding network resources. An RSU with low computational load and low communication congestion will have a final embedding vector that is significantly different from one in a "hotspot" area. This differentiated information aggregation, achieved through attention weights, is precisely the foundation for GAPO's ability to understand the global network state and to make intelligent decisions, and it also provides a transparent window for us to understand the internal working mechanism of the model.

In this way, the GAPO algorithm uses GNNs to effectively compress raw, high-dimensional, and structured network state information into low-dimensional, dense, and context-rich node embeddings, providing a high-quality state representation for subsequent reinforcement learning decisions.

4.3. Attention-Based Actor–Critic Policy Network

In the GAPO algorithm, we employ the Actor–Critic reinforcement learning framework to learn the optimal task offloading and resource allocation policy. This framework consists of two core neural networks: the Actor network and the Critic network. These two networks share the node state embeddings extracted by the underlying GNN encoder and perform their respective computations based on this foundation. This section will detail the structure and function of the Actor and Critic networks.

As shown in Figure 4, the Actor network is responsible for formulating a specific action policy based on the current environmental state and task information. For each task to be processed, the Actor network needs to output a joint action, which includes the discrete selection of an offloading destination and two continuous resource allocation ratios (offloading ratio and computational resource allocation ratio). The inputs to the Actor network are the GNN node embeddings $\hbar_u$ and the current task feature vector $\mathcal{X}_{task}$. For the discrete action of selecting an offloading destination, the Actor network employs a scoring and selection module based on an attention mechanism. First, a candidate target embedding matrix $C_{target} \in \mathbb{R}^{(1+M_{slot})\times \mathcal{D}_{emb}}$ is constructed, where M_{slot} is the number of fixed slots defined for RSUs in the observation space. The first row corresponds to the embedding of the source vehicle, $\hbar_{veh}$. The subsequent M_{slot} rows correspond to the embeddings of the various RSU slots. If an RSU slot is invalid in the current observation, its corresponding embedding can be set to a zero vector or to a special marker vector, and it will be masked out in the subsequent attention calculation. To intelligently select the offloading destination, the Actor network uses a query-based attention mechanism, which can be intuitively understood as an "intelligent matching" process. First, the model concatenates the GNN embedding of the current task's source vehicle $\hbar_{veh}$, the task features $\mathcal{X}_{task}$, and the global network state g_{stats}, and then transforms this into a query vector through a small MLP network as follows:

$$q_{att} = MLP_{query}([\hbar_{veh} \oplus \mathcal{X}_{task} \oplus g_{stats}]) \tag{21}$$

where MLP_{query} is a small multi-layer perceptron, with an output dimension typically identical to the node embedding dimension $\mathcal{D}_{emb}$, and uses a Tanh activation function, and $\oplus$ denotes vector concatenation. This query vector can be understood as a concise representation of the current decision-making needs, i.e., for this specific task, considering the current overall network environment, what characteristics should an ideal offloading destination have. The GNN embeddings of the candidate targets, $c_j \in C_{target}$, form the set of targets. As mentioned earlier, the GNN encoder has already encoded the resource status and topological context of each node into these embedding vectors. Therefore, the embedding c_j of an RSU with low load and high computational power naturally becomes a high-quality target for the aforementioned "query". The Actor network calculates an attention score e_j by computing the scaled dot-product similarity between the query vector q_{att} and each candidate answer as follows:

$$e_j = \frac{q_{att} \cdot c_j^T}{\sqrt{\mathcal{D}_{emb}}} \tag{22}$$

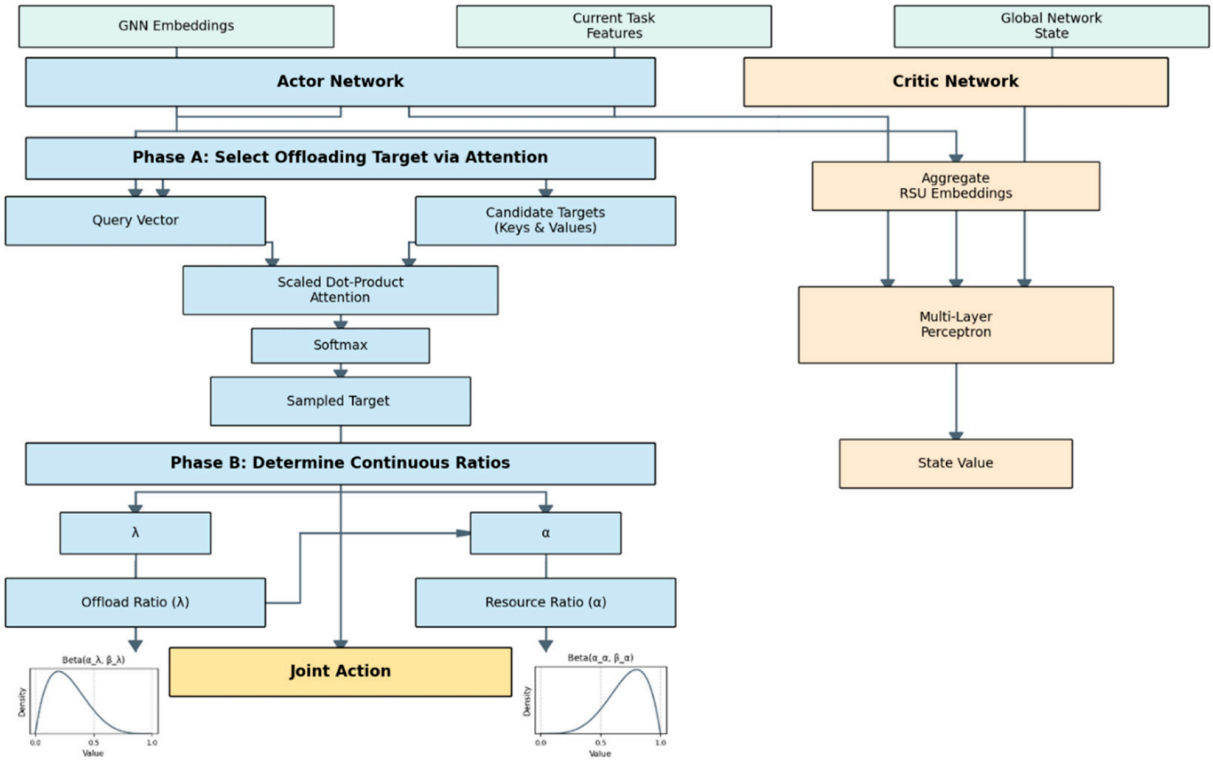

Figure 4. Attention-based Actor–Critic Policy Network.

An RSU with ideal characteristics will have its embedding vector c_j be more "aligned" or "similar" to the query vector q_{att} in the high-dimensional space, thus receiving a higher attention score. For invalid candidate targets, their attention scores e_j are set to negative infinity, so that their probability will approach zero in the subsequent Softmax operation.

The calculated attention scores are converted into a probability distribution for selecting each offloading destination, P_{target}, using a Softmax function as follows:

$$P_{target} = Softmax(e) \tag{23}$$

where $e = [e_0, e_1, \ldots, e_{M_{slot}}]$. This is a Categorical distribution, and the agent will sample a discrete action $\mathcal{K}_i \in \{0, 1, \ldots, M_{slot}\}$ from this distribution, representing the selected offloading destination (0 for local, 1 to M_{slot} for the corresponding RSU slot).

After determining the offloading destination $\mathcal{K}_i$, the Actor network needs to further decide on two continuous ratio values. For the offloading ratio, the input is the concatenated vector $[h_{veh} \oplus h_{target} \oplus \mathcal{X}_{task} \oplus g_{stats}]$. An independent MLP network maps this input to the two parameters (a_λ, b_λ) of a Beta distribution. These two parameters are typically passed through a Softplus activation function to ensure they are positive and greater than or equal to 1, to maintain the good shape of the Beta distribution. The offloading ratio λ_i is then sampled from the $Beat(a_\lambda, b_\lambda)$ distribution. For the computational resource allocation ratio, the input is the concatenated vector $[h_{veh} \oplus h_{target} \oplus \mathcal{X}_{task} \oplus g_{stats} \oplus \lambda_i]$. Here, the sampled offloading ratio λ_i is also included as an input, because the amount of computational resources to allocate may be related to how much computation is being offloaded. Another independent MLP network maps this input to the two parameters (a_α, b_α) of a Beta distribution, also processed by Softplus. The allocation ratio α_i is then sampled from the $Beat(a_\alpha, b_\alpha)$ distribution.

The final output of the Actor network is the joint action $(\mathscr{K}_i, \lambda_i, \alpha_i)$. During the training process, it is also necessary to calculate the log-probability of taking this action, $log\pi(\mathscr{K}_i, \lambda_i, \alpha_i|\mathcal{S}_t)$, which is the sum of the log-probabilities of each independent (or conditionally independent) action part, which is calculated as follows:

$$\log \pi(\mathscr{K}_i, \lambda_i, \alpha_i|\mathcal{S}_t) = \log P(\mathscr{K}_i|\mathcal{S}_t) + \log P(\lambda_i|\mathcal{S}_t, \mathscr{K}_i) + \log P(\alpha_i|\mathcal{S}_t, \mathscr{K}_i, \lambda_i) \quad (24)$$

The purpose of the Critic network is to learn a state-value function $V^\pi(\mathcal{S}_t)$, used to evaluate the expected cumulative return that can be obtained by following policy π from the current state $\mathcal{S}_t$. The output of the Critic network is used to calculate the advantage function, thereby guiding the update of the Actor network. The input to the Critic network is similar to that of the Actor network, aimed at comprehensively capturing the current state information. To obtain a fixed-size state representation, we aggregate the embeddings of all RSUs, for example, by using average pooling to obtain a single RSU node embedding, and then concatenate it with the embedding of the current task's source vehicle $\hbar_{veh}$. The task feature vector and global statistics are the same as for the Actor network. The Critic network is a multi-layer perceptron, containing several hidden layers and one output layer. The output layer of the Critic network is a single neuron, which outputs a scalar value $V(\mathcal{S}_t)$, an estimate of the value of the current state $\mathcal{S}_t$.

During the training process, the state-value estimates $V(\mathcal{S}_t)$ and $V(\mathcal{S}_{t+1})$ from the Critic network, along with the received immediate reward r_t, are used to calculate the advantage function $A(\mathcal{S}_t, \mathscr{a}_t)$ using a method like generalized advantage estimation (GAE) [21], as follows:

$$\hat{A}_t^{GAE}(\gamma, \lambda_{GAE}) = \sum_{l=0}^{\infty} (\gamma\lambda_{GAE})^l \delta_{t+l} \quad (25)$$

where $\delta_t = r_t + \gamma V(\mathcal{S}_{t+1}) - V(\mathcal{S}_t)$ is the temporal-difference error, γ is the discount factor, and λ_{GAE} is the smoothing parameter for GAE. This advantage function $\hat{A}_t^{GAE}$ measures how much better or worse performing action $\mathscr{a}_t$ is compared to the average action in state $\mathcal{S}_t$, and it is used to update the Actor network's policy.

To illustrate this process, consider the following practical example: An RSU, let us call it $\mathscr{r}_a$, has low computational load and high capacity. Its initial feature vector would reflect this. The GNN encoder, through its neighborhood aggregation layers, will create a final embedding $\hbar_a$ for $\mathscr{r}_a$. This embedding not only encodes its own favorable state but incorporates features from its neighbors. If $\mathscr{r}_a$ is connected to other uncongested RSUs via low-latency R2R links, its embedding $\hbar_a$ will be further refined to represent a highly accessible and powerful resource.

When a vehicle generates a compute-intensive task, the Actor network formulates a query vector $\mathscr{q}_{att}$ that, through training, learns to represent the "ideal" target for such a task. In the high-dimensional embedding space, the model learns to place the embeddings of desirable RSUs like $\hbar_a$ "closer" to such query vectors. The attention mechanism then computes the similarity (via a scaled dot-product) between $\mathscr{q}_{att}$ and $\hbar_a$. Because they are semantically aligned, they will produce a high attention score. By contrast, a heavily loaded $\mathscr{r}_b$ will have an embedding $\hbar_b$ that is "far" from $\mathscr{q}_{att}$, resulting in a low score. In this way, the attention mechanism intelligently identifies $\mathscr{r}_a$ as a high-quality offloading target, not just based on its own features, but on the rich, contextualized understanding of its state provided by the GNN. The entire forward pass of the Actor network, from receiving state information to generating the joint action, is summarized in Algorithm 2.

Algorithm 2: Actor Network Forward Pass for Joint Action Generation

Input : GNN node embeddings H, vehicle embedding h_{veh}, task features x_{task}, global stats g_{stats}
Output : Joint action $(\mathcal{K}_i, \lambda_i, \alpha_i)$, log probability $\log \pi(\mathcal{K}_i, \lambda_i, \alpha_i | \mathcal{S}_t)$
Phase A: Select Offloading Target via Attention
$q_{att} \leftarrow MLP_{query}([h_{veh} \oplus x_{task} \oplus g_{stats}])$
$C_{target} \leftarrow [h\_veh, h\_rsu\_1, ..., h\_rsu\_M]$
for each candidate c_j in C_{target} **do**
    $e_j \leftarrow (q_{att} \cdot c_j) / \text{sqrt}(\mathcal{D}_{emb})$
end for
Mask scores for invalid/unavailable targets
$P_{target} \leftarrow Softmax([e_0, e_1, \ldots, e_M])$
Sample target index $\mathcal{K}_i \sim \text{Categorical}(P_{target})$
$h_{target} \leftarrow C_{target}[\mathcal{K}_i]$
Phase B: Determine Continuous Ratios
if $\mathcal{K}_i = 0$ **then**
    $\lambda_i \leftarrow 0$, $\alpha_i \leftarrow 1$
else
    $a_\lambda, b_\lambda \leftarrow Softplus(MLP_\lambda([h_{veh} \oplus h_{target} \oplus x_{task} \oplus g_{stats}])) + 1$
    Sample offload ratio $\lambda_i \sim Beat(a_\lambda, b_\lambda)$
    $a_\alpha, b_\alpha \leftarrow Softplus(MLP_\alpha([h_{veh} \oplus h_{target} \oplus x_{task} \oplus g_{stats} \oplus \lambda_i])) + 1$
    Sample resource ratio $\alpha_i \sim Beat(a_\alpha, b_\alpha)$
end if
Calculate total log probability $\log \pi$ from the distributions
return $(\mathcal{K}_i, \lambda_i, \alpha_i)$, $\log \pi(\mathcal{K}_i, \lambda_i, \alpha_i | \mathcal{S}_t)$

4.4. Policy Optimization Algorithm

To effectively train the attention-based Actor–Critic policy network, the GAPO algorithm employs a deep reinforcement learning algorithm—proximal policy optimization (PPO) [19]. PPO is well-known for its excellent balance between sample efficiency, implementation simplicity, and performance stability, making it a popular choice for solving complex problems with continuous and discrete action spaces.

PPO belongs to the family of policy gradient (PG) methods which directly optimize a parameterized policy $\pi_\theta(a_t|\mathcal{S}_t)$ to maximize the expected cumulative return. Compared to traditional PG methods, PPO introduces a special "clipping" mechanism or an adaptive KL-divergence penalty to constrain the step size of each policy update. This helps to avoid drastic performance drops caused by a single large update, thereby improving training stability.

In GAPO, we use the clipped surrogate objective function version of PPO. Its core idea is to ensure that the new policy does not deviate too far from the old policy during each update.

As described in Section 4.3, the Actor network outputs the parameters of the probability distribution for the joint action $(\mathcal{K}_i, \lambda_i, \alpha_i)$. When interacting with the environment to collect data, actions are sampled from these distributions according to the current policy. The Critic network learns a state-value function, which is used to evaluate the expected cumulative return of following the current policy from state $\mathcal{S}_t$.

The Actor network is updated by maximizing the following clipped surrogate objective function $L^{CLIP}(\theta')$ as follows:

$$L^{CLIP}(\theta') = \mathbb{E}_t\left[min\left(\frac{\pi_{\theta'}(a_t|\mathcal{S}_t)}{\pi_{\theta}(a_t|\mathcal{S}_t)}\right)\hat{A}_t^{\pi_\theta}, clip\left(\frac{\pi_{\theta'}(a_t|\mathcal{S}_t)}{\pi_{\theta}(a_t|\mathcal{S}_t)}, 1-\epsilon_{clip}, 1+\epsilon_{clip}\right)\hat{A}_t^{\pi_{\theta'}}\right] \quad (26)$$

where $\pi_{\theta'}$ is the current policy, $\hat{A}_t^{\pi_{\theta'}}$ is the GAE advantage estimate calculated from data collected under the old policy π_θ; the $clip(x, low, high)$ function clips x into the range $[low, high]$; and ϵ_{clip} is a small hyperparameter that defines the clipping range.

This objective function limits the magnitude of the policy update by taking the minimum of two terms. If the probability ratio $\frac{\pi_{\theta'}(a_t|\mathcal{S}_t)}{\pi_{\theta}(a_t|\mathcal{S}_t)}$ would lead to a large increase (when the advantage is positive) or a large decrease (when the advantage is negative) in the objective function, the clipping mechanism takes effect, preventing the policy from updating too quickly.

The Critic network is updated by minimizing the mean squared error (MSE) between its predicted state value and the actual return. The target return is to minimize the difference with the Monte Carlo return as follows:

$$L^{VF}(\phi_{critic}) = \mathbb{E}_t\left[\left(V_{\phi_{critic}}(\mathcal{S}_t) - R_t^{target}\right)^2\right] \quad (27)$$

where R_t^{target} is an estimate of the true value of state $\mathcal{S}_t$.

The critical link between these mathematical updates and the physical reality of traffic load capacity is established through two core mechanisms: the reward function and the GNN-based state representation. Our multi-objective reward function is explicitly designed to penalize undesirable outcomes. When an action leads to a decision that causes high task latency or sends traffic over a congested link, the agent receives a low or negative reward. During the update process, via Equation (26), this low reward translates into a low or negative advantage estimate. Consequently, the PPO algorithm will update the Actor's policy to reduce the probability of taking such a detrimental action in similar states in the future. The GNN encoder provides the necessary context for this learning to be effective. It processes the raw network state, including the ρ_{link} for every communication link. When a link is congested, this high utilization value is encoded into the node embeddings. The Critic network, by observing these embeddings, learns to associate congested network states with lower expected future rewards, producing a more accurate value estimate. The Actor, in turn, receives these rich, congestion-aware state representations and learns a more nuanced policy that can distinguish between a high-capacity free link and a high-capacity but already saturated link. In essence, when the traffic load approaches or exceeds capacity, the environment provides a "punishment" signal via the reward. The GNN ensures the agent "sees" the congestion, and the Actor–Critic update mechanism is the process by which the agent learns from this punishment to make better, congestion-aware decisions.

The total loss function for the Actor and Critic networks is a weighted sum of their respective losses, which is calculated as follows:

$$L^{TOTAL}(\theta', \phi_{critic}) = L^{CLIP}(\theta') + c_{vf} \cdot L^{VF}(\phi_{critic}) \quad (28)$$

This total loss function is optimized via stochastic gradient descent.

Through the stable learning process of the PPO algorithm, GAPO can effectively optimize its complex policy network, which includes GNNs and attention mechanisms, thereby learning a high-performance multi-objective task offloading and resource allocation policy in a multi-hop VEC environment.

4.5. Convergence and Complexity Analysis

The core of the GAPO algorithm is a deep reinforcement learning framework based on proximal policy optimization (PPO). The convergence of PPO has been theoretically proven and extensively validated in practice for standard Markov decision processes (MDPs). Our analysis aims to connect these theoretical foundations to our complex multi-hop VEC environment.

The task offloading problem in vehicular edge computing can be modeled as a large-scale MDP, defined by the tuple (S, A, P, R, γ). The state space S is represented by the dynamic graph $\mathscr{G}_t$, which includes all nodes and their attributes. Due to vehicle mobility and random task arrivals, this is a high-dimensional, continuous, and dynamically changing state space. Our proposed GNN encoder aims to learn a low-dimensional and informative state representation $s_t = \mathbb{F}_{GNN}(\mathscr{G}_t)$, which is intended to approximate the Markov property, i.e., $P(s_{t+1}|s_t, a_t) \approx P(s_{t+1}|s_t, a_t, \ldots, s_0, a_0)$. The action space A is a hybrid action space containing the discrete offloading destination selection $\mathscr{K}_i$ and the continuous resource allocation ratios (λ_i, α_i). The transition probability P is determined by the dynamics of the VEC environment. As described in Section 3.4, the reward function R is a multi-objective weighted reward function combining latency and network congestion.

Due to the high dimensionality and dynamics of the state space and the hybrid nature of the action space, traditional tabular reinforcement learning methods are not feasible. We must use function approximators. We use an Actor network with parameters θ and a Critic network with parameters ϕ to approximate the optimal policy and state-value function, respectively. Function approximation introduces errors, which is a core issue that must be addressed in convergence analysis.

Our goal is to prove that by optimizing the PPO clipped surrogate objective function, GAPO's policy parameters θ can converge, causing the policy performance $\mathscr{J}(\theta) = \mathbb{E}_{\gamma\sim\pi_\theta}\left[\sum_{t=0}^{\mathscr{T}} \gamma^t R(S_t, A_t)\right]$ to reach a local optimum. To construct the proof, we assume that the GNN encoder and the Critic network have a bounded approximation error for the true value function $V^*(S_t)$. That is, there exists a small constant ε_V as follows:

$$SUP_{S_t\in S}\left|V^*(S_t) - V_\phi(\mathbb{F}_{GNN}(\mathscr{G}_t))\right| \leq \varepsilon_V \tag{29}$$

This assumption indicates that our GNN can effectively extract key state information. Additionally, since the reward function and value function outputs are bounded, the advantage function estimate has an upper bound A_{max}.

Lemma 1. *For any two policies π_θ and $\pi_{\theta'}$, their performance difference can be expressed as follows:*

$$\mathscr{J}(\theta') - \mathscr{J}(\theta) = \mathbb{E}_{\gamma\sim\pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(S_t, a_t)\right] \tag{30}$$

where $A^{\pi_\theta}(S_t, a_t)$ is the true advantage function under the old policy π_θ.

Directly optimizing the expression in Lemma 1 is difficult because its expectation is based on the unknown future policy $\pi_{\theta'}$. PPO addresses this by introducing a surrogate objective function that is a lower bound of the original objective within a neighborhood of θ'. We use the clipped surrogate objective function $L^{CLIP}(\theta')$ described in Section 4.4.

Theorem 1. *Performing one gradient ascent update on $L^{CLIP}(\theta')$ can guarantee an approximate monotonic improvement in policy performance. Specifically, there exists a term $\delta_k > 0$, which depends on function approximation errors and sampling errors, as follows:*

$$\mathscr{J}(\theta_{k+1}) \geq \mathscr{J}(\theta_k) - \delta_k \tag{31}$$

Proof of Theorem 1. According to the analysis of PPO by Schulman et al. [22], a lower bound on the performance improvement can be established. □

Theorem 2. *The performance sequence* $\{\mathcal{J}(\theta_k)\}_{k=0}^{\infty}$ *corresponding to the policy sequence* $\{\pi_{\theta_k}\}_{k=0}^{\infty}$ *generated by the GAPO algorithm converges, and its policy parameter gradient converges to zero, which is calculated as follows:*

$$\lim_{k\to\infty} \|\nabla_\theta \mathcal{J}(\theta_k)\| = 0 \tag{32}$$

Proof of Theorem 2. Since the reward function R has an upper bound R_{max}, the total expected return $\mathcal{J}(\theta)$ also has an upper bound, as follows:

$$\mathcal{J}(\theta) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma} \tag{33}$$

According to Theorem 1, the policy performance sequence $\{\mathcal{J}(\theta_k)\}$ is approximately monotonically non-decreasing. A real-valued sequence that is approximately monotonically non-decreasing and has an upper bound must converge. Therefore, $\lim_{k\to\infty} \mathcal{J}(\theta_k)$ exists. Since the performance sequence converges, the difference between its adjacent terms must approach zero. According to policy gradient theory and the PPO update mechanism, performance improvement is related to the norm of the policy gradient. When performance no longer improves, it indicates that the policy parameter updates have stabilized, i.e., the gradient $\nabla_\theta \mathcal{J}(\theta_k)$ approaches 0. Therefore, the policy parameters of the GAPO algorithm will converge to a stationary point, which corresponds to a locally optimal task offloading and resource allocation policy.

Regarding the computational complexity of the GAPO algorithm during a single offloading decision, we mainly focus on the inference phase complexity, as it directly affects the system's real-time response capability. Let the system have N vehicles and M RSUs, so the total number of nodes is $N_t = N + M$, and the total number of edges is E_t. The GNN embedding dimension is $\mathcal{D}_{emb}$. The complexity of the GNN is primarily determined by message passing. For one GAT layer, its complexity is $O(N_t \mathcal{D}_{in} \mathcal{D}_{out} + E_t \mathcal{D}_{out})$. For an L-layer GNN, the total complexity is approximately $O(L \cdot E_t \cdot \mathcal{D}_{emb})$, as the number of edges is typically much larger than the number of nodes. The query vector q_{att} in the Actor network is generated by a small MLP, with a complexity that can be considered $O(1)$. The attention score calculation involves the dot product of the query vector with M + 1 candidate target embeddings, with a complexity of $O(M \cdot \mathcal{D}_{emb})$. The two continuous actions are generated by two small MLPs, with complexity $O(1)$. In summary, the total complexity of a single decision is dominated by the GNN encoder, which is $O(N_t \mathcal{D}_{in} \mathcal{D}_{out} + E_t \mathcal{D}_{out})$. Since the network topology is sparse, E_t has a linear relationship with N_t. Therefore, the complexity can be approximated as $O(N_t \cdot \mathcal{D}_{emb})$. The computational complexity of GAPO is polynomial with respect to the network size, making it practically applicable in dynamic VEC environments. □

5. Experimental Analysis

To systematically evaluate the performance of the proposed GAPO algorithm in dynamic multi-hop VEC environments, this chapter designs a series of comparative and ablation experiments. We first introduce the experimental environment, parameter configurations, and training hyperparameters. Then, we provide a detailed description of the baseline algorithms used for comparison. Finally, by presenting and analyzing simulation

result charts, we conduct an in-depth analysis of the performance of GAPO and other algorithms under different scenarios, verifying its superiority in minimizing task latency and alleviating R2R link congestion.

5.1. Experimental Setup

We conduct our experiments in a simulated unidirectional multi-lane highway scenario. This scenario consists of a series of dynamically moving vehicles and fixedly deployed roadside units. Vehicles randomly generate computational tasks and make decisions to either process them locally or offload them to one or more RSUs. The key parameters of the simulation environment are listed in Table 2. These parameters constitute the baseline scenario for evaluating the performance of our algorithm. The key hyperparameters for the training process are shown in Table 3.

Table 3. GAPO Training Hyperparameters.

Hyperparameter	Value
Learning Rate	1×10^{-4}
Discount Factor (γ)	0.99
GAE Smoothing Parameter (λ)	0.95
Training Epochs per Update Cycle	10
Experience Replay Buffer Size	2048
Mini-batch Size for Updates	4
Entropy Loss Coefficient	0.01
Value Function Loss Coefficient	0.5
GNN Output Node Embedding Dimension	32
GAT Attention Head Count	2
MLP Hidden Layer Dimension in the Actor Network	64
MLP Hidden Layer Dimension in the Critic Network	64

The selection of key hyperparameters is critical to the performance and stability of the GAPO algorithm. Below, we provide the rationale for our choices, which are based on the established literature and empirical observations from our experiments. The output dimension of the GNN encoder determines the richness of the learned node representations. A dimension of 32 was chosen to strike a balance between model expressiveness and computational cost. It is large enough to capture complex node features and topological context but small enough to prevent overfitting and to ensure that the GNN encoder can operate efficiently for timely decision-making. Our ablation studies (Section 5.3) show that this dimension, when combined with the full model, yields excellent performance, confirming its adequacy. Following the original GATv2 architecture [20], we employ a multi-head attention mechanism. Using multiple heads allows the model to learn different weighted aggregations of neighbor information in parallel, capturing more diverse relationship patterns and stabilizing the learning process. This is particularly useful in our heterogeneous VEC environment, where the importance of a neighbor may depend on various factors. The learning rate is a crucial parameter for training stability. We selected a relatively conservative value of 1×10^{-4}, which is a common and robust choice for PPO with the Adam optimizer. This smaller learning rate helps prevent destructive policy updates, especially given the complexity of our model and the dynamic nature of the environment. The stable convergence shown in our training curves validates that this rate facilitates effective learning without policy collapse.

5.2. Comparative Experiments

To comprehensively evaluate the performance of GAPO, we selected several representative baseline algorithms for comparison, covering traditional strategies, heuristic methods, and other reinforcement learning approaches.

- Local-Only: All tasks generated by vehicles are processed on their local on-board units, without any network offloading. This strategy completely avoids network communication latency and congestion but is limited by the finite computational power of the vehicles.
- Random Offload: A simple baseline strategy where vehicles make decisions randomly. It randomly selects an available RSU as the offloading target or chooses to process the task locally. If offloaded, the task is fully offloaded.
- Greedy Lowest Load: A heuristic algorithm where, at the time of decision, the vehicle evaluates the expected computational load of all reachable computation nodes, including itself. It always greedily selects the node with the lowest computational load as the offloading target.
- A2C (Advantage Actor–Critic) [23]: A classic reinforcement learning algorithm. To adapt it to the complex environment of this study, we designed a manual feature engineering module for it. This module transforms the graph-structured observation into a fixed-length vector. Specifically, it extracts a series of aggregated features for the current task, the source vehicle, and each RSU, such as direct connectivity with the source vehicle, the number of hops and estimated latency of the multi-hop path to the RSU, and the RSU's own computational load. These features are concatenated into a long vector and fed into a standard MLP-based Actor–Critic network. The performance of the A2C algorithm is highly dependent on the completeness and effectiveness of these hand-crafted features, which stands in stark contrast to GAPO's core idea of automatically learning network state representations via GNN.

To evaluate the performance of GAPO, we designed five different scenarios focusing on three core dimensions: network scalability, resource and load adaptability, and environmental heterogeneity. We tested each algorithm for 50 episodes in each scenario. We primarily focus on two key metrics: Average Task Latency and Average R2R Link Congestion. The latter measures the degree of congestion on the R2R backbone links caused by offloading traffic, calculated as the average of the actual utilization exceeding the stability threshold φ across all R2R links involved in task transmissions.

Given that the task offloading problem is NP-hard, assessing the scalability of any proposed algorithm is of paramount importance. As the problem size grows, the solution space expands exponentially, making it intractable for exact methods and posing a significant challenge for heuristics and learning-based approaches. Figure 5 directly evaluates the scalability of all tested algorithms by increasing the number of vehicles from 40 to 80, which proportionally increases the system load and resource competition. The results reveal a clear distinction in scalability. The performance of both the greedy lowest load and the A2C algorithms deteriorates significantly as the network scale increases. The greedy algorithm's myopic nature becomes its downfall at scale; with more vehicles making decisions, the probability of creating traffic "hotspots" at the computationally strongest RSUs increases, leading to disproportionately severe R2R link congestion. Similarly, the classic A2C algorithm, which relies on a flattened, fixed-size state vector, suffers from an information bottleneck. As the network grows, this vector becomes increasingly inadequate at capturing the complex, expanding web of topological relationships, resulting in a sharp decline in decision quality. By contrast, GAPO demonstrates superior scalability, a direct result of its architectural design. The core of this advantage lies in the GNN encoder. GNNs process information locally through message passing, meaning the computation

for a node's embedding depends on the size of its immediate neighborhood, not the total number of nodes in the graph. This inherent locality prevents the "curse of dimensionality" that plagues the A2C model. Consequently, as the network scales, GAPO can efficiently and effectively continue to generate high-quality, context-aware state representations. This allows it to maintain low latency and minimal congestion, proving its viability as a practical solution for large-scale instances of this NP-hard problem.

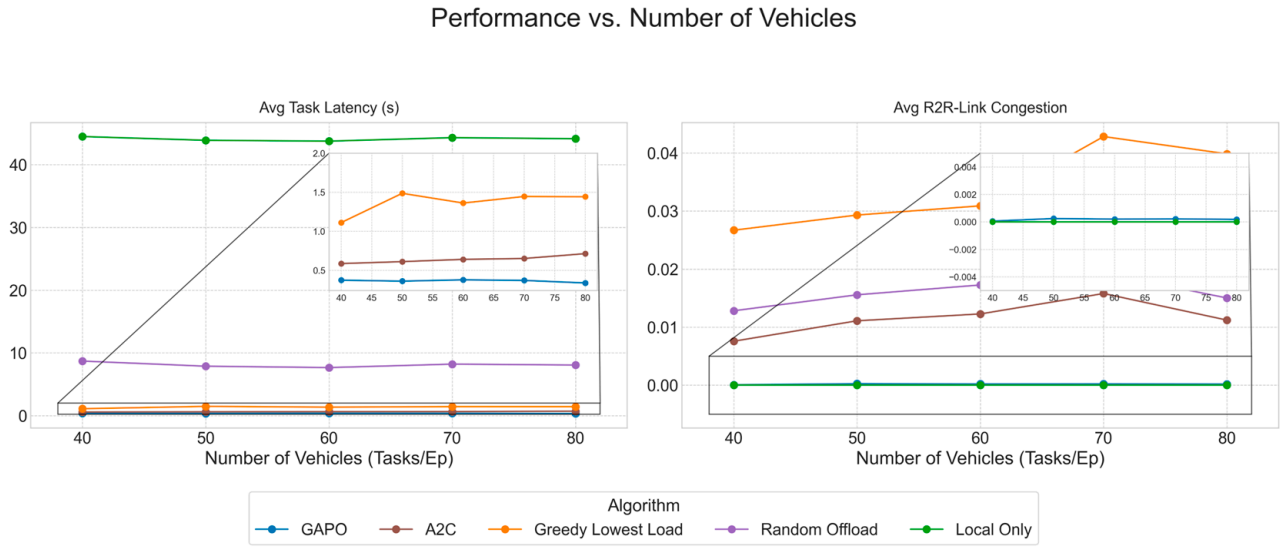

Figure 5. Impact of Vehicle Count on Performance.

We evaluated resource adaptability by adjusting the computational capacity of all RSUs with a scaling factor. As shown in Figure 6, when the RSU computational capacity increases, the latency of all offloading algorithms decreases. GAPO leads significantly with the lowest latency and R2R congestion. It is worth noting that, even when RSU capacity is low (factor of 0.5), GAPO still maintains extremely low latency, demonstrating its ability to efficiently utilize limited computational resources. Although the runner-up algorithm, A2C, also benefits from the capacity increase, its performance improvement is not as significant as GAPO's, and it is consistently accompanied by higher R2R congestion, indicating its failure to effectively co-optimize computation and communication resources.

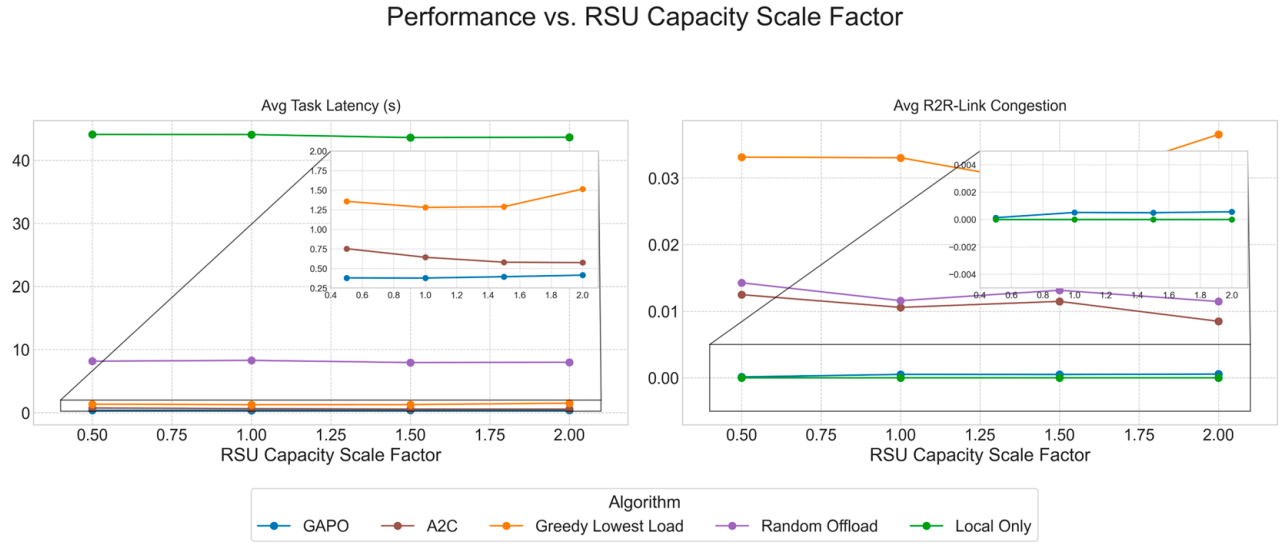

Figure 6. Impact of RSU Computational Capacity on Performance.

We tested system load adaptability by varying the task Inter-Arrival Time (IAT). In Figure 7, when the system enters the high-load, extreme-pressure scenario (IAT = 0.02 s), the performance gap between the algorithms widens dramatically. GAPO still maintains the best performance, with its latency and R2R congestion increasing much less than the other algorithms. This highlights GAPO's adaptability in highly dynamic and competitive environments. Its core advantage lies in the fact that, when the entire network becomes congested, the global contextual information provided by the GNN becomes crucial. By contrast, A2C's fixed feature combination can no longer effectively represent the complex network state under high load, leading to a drop in decision quality. GAPO is able to discover non-greedy but globally superior offloading paths, such as choosing a slightly closer but slightly higher-load RSU to avoid the backbone link paralysis caused by excessive traffic concentration on a few optimal RSUs.

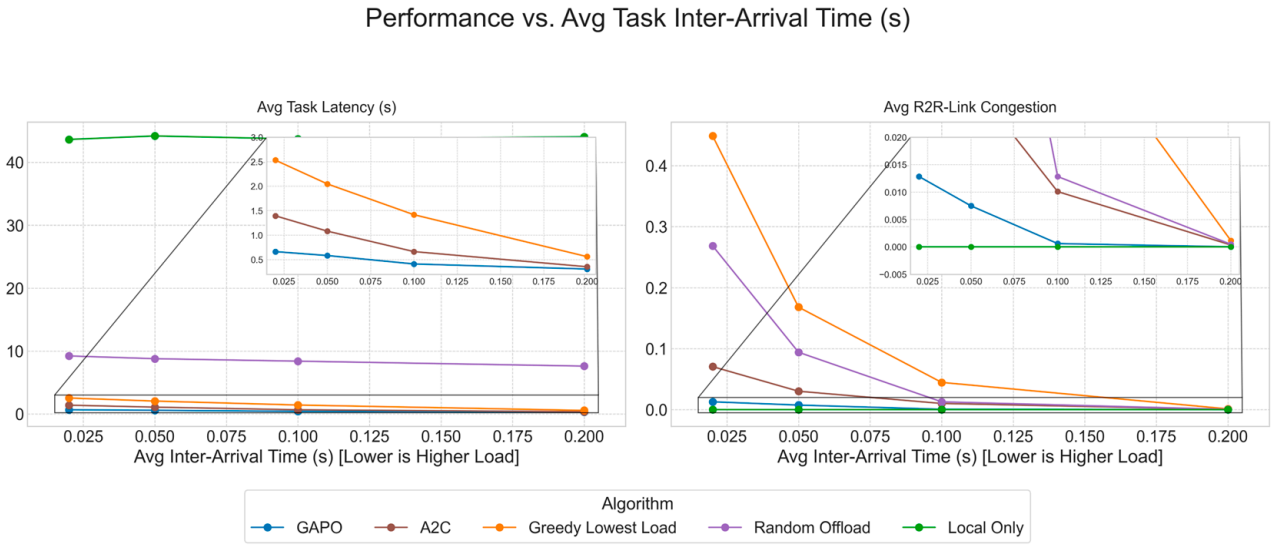

Figure 7. Impact of Task Arrival Interval on Performance.

Real-world VEC environments are often heterogeneous. We set up two scenarios, task heterogeneity and RSU computational resource heterogeneity, to evaluate the algorithm's performance under environmental heterogeneity. In the task heterogeneity scenario, we tested three task types: balanced, compute-intensive, and data-intensive. The balanced type uses the default task data and computation sizes. The compute-intensive type maintains the default data size but increases the computation size to (1.0, 2.5) Gc. The data-intensive type maintains the default computation size but increases the data size to (800, 1024) KB. The experimental results are shown in Figure 8. For compute-intensive tasks, the latency of Local Only skyrockets to about 76 s, highlighting the bottleneck of the vehicle's local computational capacity. GAPO still performs optimally, proving its ability to intelligently match heavy computation tasks to the most suitable RSU resources. For data-intensive tasks, which require large amounts of data transmission, the demands on network bandwidth are higher. In this case, A2C's R2R link congestion rate increases significantly compared to the balanced type, indicating that classic reinforcement learning algorithms struggle to fully capture the global network topology and inter-node dependencies.

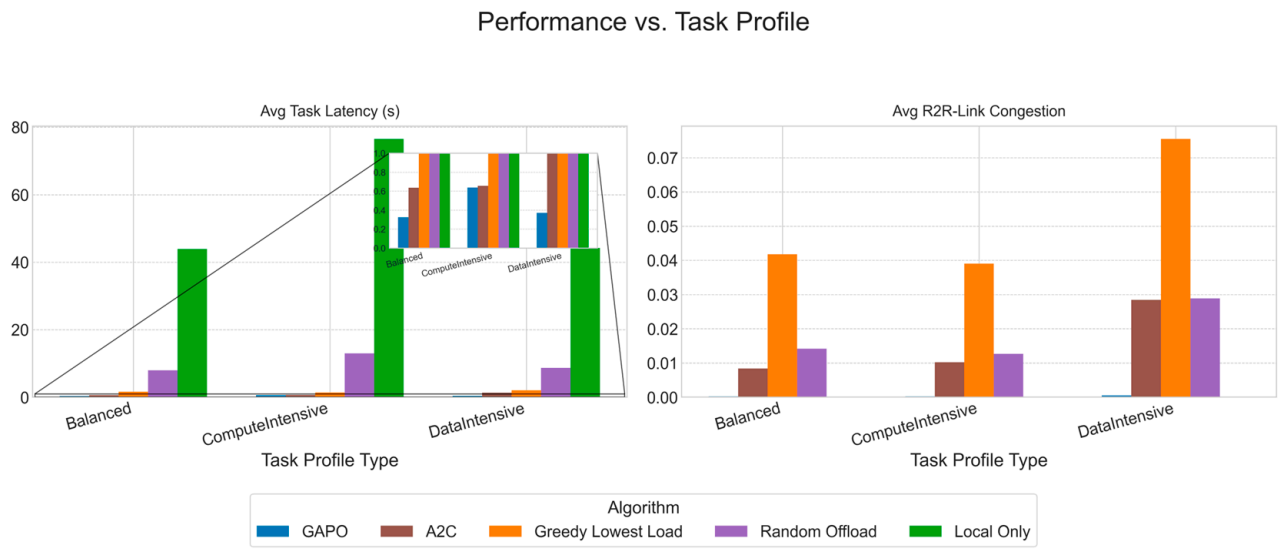

Figure 8. Impact of Task Type Heterogeneity on Performance.

To evaluate the impact of RSU computational resource heterogeneity, we set up "Skewed" and "Default" heterogeneity modes. In the "Skewed" mode, the RSU compute capacities are set to [10.0, 10.0, 15.0, 30.0, 40.0] GHz. In the "Default" mode, they are [15.0, 20.0, 25.0, 30.0, 35.0] GHz. As shown in Figure 9, in the "Skewed" scenario, a few RSUs possess far superior computational power than others. This imbalanced resource distribution poses a severe challenge to load balancing strategies. The experimental results show that greedy lowest load and A2C perform poorly in this scenario, leading to high R2R congestion. They tend to offload a large number of tasks to the few "super" RSUs, causing severe congestion on the paths to these RSUs and creating a "hotspot" effect. By contrast, by leveraging its awareness of global topology and load, GAPO successfully avoids this trap. It understands that diverting traffic to some slightly weaker but more accessible RSUs can achieve lower overall latency and better system stability, thus realizing true global load balancing.

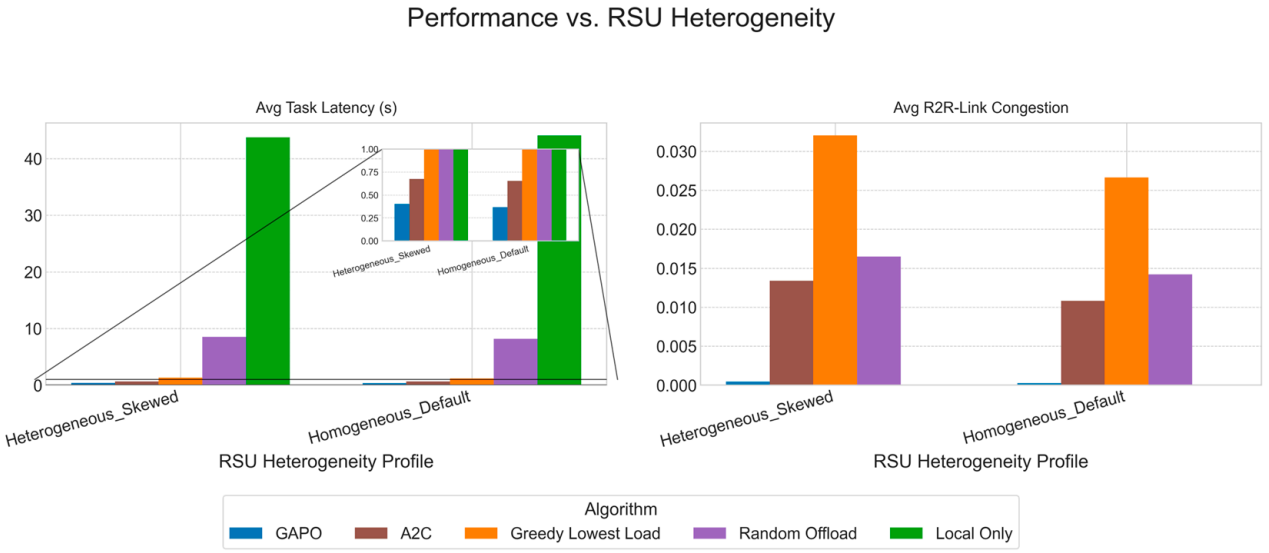

Figure 9. Impact of RSU Computational Resource Heterogeneity on Performance.

The substantial performance disparities observed across the algorithms in Figures 5–7 are not arbitrary; they directly reflect the fundamental differences in their underlying logic and awareness of the network state. The Local Only strategy, by design, serves as a stark baseline, consistently showing the highest latency because it completely forgoes the benefits of edge computing. It forces all tasks onto the vehicle's computationally weak on-board unit, creating an immediate and severe processing bottleneck that underscores the critical need for an effective offloading mechanism. The greedy lowest load algorithm represents a step toward utilizing edge resources, yet its performance is severely hampered by a myopic and communication-blind approach. Its sole focus on instantaneous computational load causes it to overlook the substantial communication latency required to reach a distant, albeit computationally less loaded, RSU. This flawed logic not only incurs high transmission delays but systematically creates network hotspots. As multiple vehicles greedily select the same "optimal" server, they funnel traffic onto the same backbone paths, leading to the severe R2R link congestion consistently observed in our results. The algorithm's attempt to find the least loaded server paradoxically leads to systemic congestion and high end-to-end latency. In stark contrast, GAPO's superior performance is a direct result of its ability to achieve a holistic, globally-aware optimization. Through its GNN encoder, GAPO builds a rich, contextualized representation of the entire network graph, where each node's embedding reflects not only its own load but the congestion state of its connecting links and neighbors. Armed with this global perspective, the DRL agent, guided by a multi-objective reward function, learns a far more sophisticated policy than a simple greedy choice. It learns to navigate the inherent trade-off between computation and communication, understanding that offloading to a slightly busier but closer RSU is often vastly superior to choosing a distant, idle server over a congested path. By effectively co-optimizing both latency and congestion, GAPO avoids the creation of hotspots and consistently achieves the lowest true end-to-end task completion time, thus explaining its significant and robust advantage across all tested scenarios.

In summary, the comprehensive comparison across multiple dimensions consistently demonstrates the significant superiority of the GAPO algorithm. Its core advantage lies in the powerful network state representation capability endowed by the GNN and the intelligent decision-making ability driven by the attention mechanism. This enables GAPO to co-optimize the two key objectives of latency and congestion in complex, dynamic multi-hop VEC environments, providing an efficient and robust task offloading solution that surpasses traditional methods and other DRL algorithms.

5.3. Ablation Study

To further validate the effectiveness of the graph neural network encoder and the attention-based Actor network in the GAPO algorithm, we conducted a series of ablation studies. By comparing with two key simplified variants, we can quantify the contribution of each component to the overall performance. We designed the following two ablation variants and compared them with the full GAPO model:

1. GAPO: The complete algorithm proposed in this paper, using both the GNN encoder and the query-based attention mechanism for target selection.
2. noAtt: This variant removes the query-based attention module from the Actor network. Instead, it directly concatenates the source vehicle's embedding, each candidate target's embedding, and the task features, then uses a simple MLP network to score each candidate, finally selecting a target via Softmax. This variant still uses the GNN encoder to obtain node embeddings but loses the ability to intelligently match task needs with target characteristics.

3. noGNN: This variant replaces the GNN encoder with a standard multi-layer perceptron (MLP). In this model, the features of each node are input into the MLP independently, completely ignoring the network topology. Therefore, this model cannot learn the contextual information of nodes through neighbor aggregation, and its state-aware capability degenerates to be similar to traditional flattened state vector methods.

We trained and tested these three models in the baseline scenario. The final performance metrics are summarized in Table 4.

Table 4. Ablation Study Results.

Model	Average Task Latency (s)	Average R2R Link Congestion Rate
GAPO	**0.374 (±0.178)**	**0.05%**
noAtt	0.477 (±0.197)	0.09%
noGNN	0.681 (±0.264)	1.64%

The data in Table 4 leads to a clear conclusion. The noGNN variant performs the worst, with its average latency being 82.1% higher than the full GAPO model, and its R2R link congestion being nearly 30 times higher. This is not merely a quantitative difference but reveals a fundamental flaw. The root cause lies in its inability to perceive network topology. The noGNN variant, which uses a standard MLP, processes each node's feature vector in isolation. It can see the attributes of individual RSUs but is completely blind to the state of the communication links connecting them.

Consequently, the noGNN agent cannot grasp the cumulative effect of multi-hop link loads. It might identify an RSU that is computationally powerful and has low local load, but it is unaware that the two- or three-hop path to reach it is severely congested. This leads the agent to make myopic, greedy decisions, repeatedly attempting to offload tasks to the most powerful RSUs on paper. This behavior creates severe network "hotspots", funneling traffic onto a few backbone links and causing the massive R2R congestion we observe. This result strongly proves that the global network topology and contextual awareness provided by the GNN are the cornerstone of achieving efficient task offloading.

The performance of the noAtt variant lies between the other two, with its average latency being 27.5% higher than the full GAPO model. This indicates that, even with the high-quality state representation provided by the GNN, a simple and coarse decision-making mechanism (concatenation and MLP scoring) cannot fully utilize this information. The query-based attention mechanism is crucial for achieving more fine-grained target selection by dynamically matching task requirements with candidate target characteristics, thereby further optimizing latency and congestion.

To further investigate the impact of these components on the learning process, we plotted and analyzed the key metrics during model training, as shown in Figure 10. Subplot (a) shows the trend of policy entropy. Entropy measures the degree of exploration or uncertainty in the policy. The entropy of all models decreases during training, indicating that the agents are gradually transitioning from exploration to exploitation, learning more deterministic policies. Among them, the entropy of noGNN decreases the slowest and has the highest final value, indicating that it had difficulty finding an effective policy and consistently maintained a high level of uncertainty. Subplots (b) and (d) show the Actor and Critic losses, respectively. The policy loss and value loss of noGNN are not only the highest in value but are extremely volatile. This reveals a deeper problem: because the noGNN variant cannot accurately perceive the network state, its Critic network fails to produce reliable state-value estimates. This results in noisy advantage signals, which in turn makes the Actor's policy updates difficult and unstable. By contrast, the loss curves of GAPO and noAtt are lower and more stable, demonstrating that the state representation

provided by the GNN greatly improves the stability and efficiency of learning. Among them, the full GAPO model has the lowest loss, indicating that the attention mechanism also plays a positive role in stabilizing the learning process.

Figure 10. Ablation Model Training Metrics. (**a**) Policy Entropy. (**b**) Actor Network Policy Loss. (**c**) Total Loss. (**d**) Critic Network Policy Loss.

In summary, the comprehensive comparison across multiple dimensions consistently demonstrates the significant superiority of the GAPO algorithm. The experimental results show that GAPO consistently outperforms traditional and heuristic-based algorithms. Its core advantage lies in the synergy between the GNN encoder and the DRL policy network. While GNN-informed heuristic methods, such as those relying on shortest-path routing over predicted link weights [24], can improve upon context-agnostic approaches, our DRL-based policy is capable of learning more complex and robust strategies. It can dynamically balance immediate costs with potential future congestion, achieving a superior trade-off between task latency and network stability.

6. Discussion

Our current experimental evaluation is based on a multi-hop highway scenario, which models a linear network topology. A natural and important question is how the proposed GAPO framework would generalize to more complex and realistic urban scenarios, such as multi-intersection grids.

The core design of GAPO is fundamentally based on a graph representation of the network, which makes it inherently suitable for more complex topologies. Unlike methods tailored for linear arrangements, our use of GNNs is topology-agnostic and can naturally process arbitrary graph structures, such as the mesh-like connectivity found in urban grids. However, transitioning to such a scenario would introduce several challenges and require specific, yet feasible, adaptations.

1. State Representation and Scalability: An urban grid would contain a larger number of RSU nodes and a more complex edge set. This increases the scale of the input graph. GAPO's GNN architecture is well-equipped to handle this. By design, GNNs like GATv2 operate on local neighborhoods. The computational complexity for updating a single node's embedding scales with its number of neighbors, not the total number of nodes in the graph. This inherent locality ensures that the state representation module of GAPO remains scalable even in larger, denser network environments.
2. Multi-Hop Pathfinding: The pathfinding algorithm in our current implementation implicitly assumes a linear chain of RSUs. For an urban grid, this module would need to be replaced with a more general graph traversal algorithm. This is a straightforward modification. The RSU network can be treated as a subgraph, where the edges are dynamically weighted by the real-time communication latency calculated by our model. A standard algorithm like Dijkstra's could then be directly applied to find the latency-optimal path between any two RSU nodes. This would allow GAPO to identify the most efficient multi-hop communication routes through the urban mesh network without changing the core RL decision-making logic.
3. V2V Communication at Intersections: Urban intersections present rich opportunities for multi-hop V2V relaying, which is not a primary focus in our highway model. Extending GAPO to fully leverage this would be a promising direction for future research. This could involve dynamically including high-value vehicle nodes into the pathfinding subgraph, further enriching the offloading options.

In conclusion, while a full empirical evaluation on an urban grid is a direction for future work, we argue that the fundamental design of GAPO is robust and generalizable. The necessary adaptations for handling complex topologies are well-defined and rely on standard algorithmic components, demonstrating a clear and feasible path for applying GAPO to more complex and realistic vehicular networks.

7. Conclusions

This paper addresses a series of challenges in task offloading decision-making within dynamic multi-hop Vehicular Edge Computing networks, including the difficulty of real-time network state perception and the complexity of the decision space. We have proposed an intelligent task offloading algorithm based on a graph attention policy optimization framework—GAPO. The core innovation of this algorithm lies in the deep integration of the powerful state-aware capabilities of graph neural networks with the sequential decision-making abilities of deep reinforcement learning. By modeling the VEC network as a dynamic attributed graph, the GNN is able to extract global, context-rich feature embeddings from complex topological structures and node states. Building on this foundation, an Actor–Critic framework based on an attention mechanism can intelligently select the optimal offloading destination according to the specific needs of the current task and jointly determine the offloading ratio and computational resource allocation. Finally, a multi-objective reward function guides the agent to learn a near-optimal policy that balances various performance indicators. Our research has designed and implemented an efficient, adaptive, and distributed task offloading framework, GAPO, providing an effective solution to the resource management problems in complex and dynamic VEC environments. While the comprehensive simulation results robustly demonstrate the effectiveness of GAPO, we acknowledge the limitations inherent in a simulation-based study. The complexities of real-world wireless channels, unpredictable vehicle mobility patterns, and diverse application workloads can introduce challenges not fully captured by even high-fidelity models. Therefore, a crucial direction for future work is to validate the performance of GAPO in a real-world hardware testbed or a large-scale, trace-driven

emulation environment. Such an endeavor would not only confirm the practical viability of our approach but provide valuable insights for bridging the gap between algorithmic innovation and real-world deployment.

Author Contributions: Conceptualization, H.Z.; methodology, X.L.; software, X.L.; validation, X.L., C.L. and L.Y.; formal analysis, X.L.; investigation, X.L.; resources, H.Z.; data curation, X.L.; writing—original draft preparation, X.L.; writing—review and editing, H.Z.; visualization, X.L.; supervision, H.Z.; project administration, H.Z.; funding acquisition, H.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was supported by the National Natural Science Foundation of China (42471213) and the Northeast Geoscience and Technology Innovation Center Regional Innovation Fund Project (QCJJ2023-49).

Data Availability Statement: The datasets presented in this article are not readily available because the data are part of an ongoing study. Requests to access the datasets should be directed to corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhang, J.; Letaief, K.B. Mobile Edge Intelligence and Computing for the Internet of Vehicles. *Proc. IEEE* **2019**, *108*, 246–261. [CrossRef]
2. Mach, P.; Becvar, Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
3. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358. [CrossRef]
4. Filali, A.; Abouaomar, A.; Cherkaoui, S.; Kobbane, A.; Guizani, M. Multi-Access Edge Computing: A Survey. *IEEE Access* **2020**, *8*, 197017–197046. [CrossRef]
5. Dai, Y.; Xu, D.; Maharjan, S.; Zhang, Y. Joint Load Balancing and Offloading in Vehicular Edge Computing and Networks. *IEEE Internet Things J.* **2018**, *6*, 4377–4387. [CrossRef]
6. Sahni, Y.; Cao, J.; Yang, L.; Ji, Y. Multi-Hop Multi-Task Partial Computation Offloading in Collaborative Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 1133–1145. [CrossRef]
7. Han, X.; Gao, G.; Ning, L.; Wang, Y.; Zhang, Y. A semidefinite relaxation approach for the offloading problem in edge computing. *Comput. Electr. Eng.* **2022**, *98*, 107728. [CrossRef]
8. Li, Y.; Zhang, X.; Sun, Y.; Wang, W.; Lei, B. Spatiotemporal Non-Uniformity-Aware Online Task Scheduling in Collaborative Edge Computing for Industrial Internet of Things. *IEEE Trans. Mob. Comput.* **2025**, 1–18. [CrossRef]
9. Ke, Q.; Silka, J.; Wieczorek, M.; Bai, Z.; Wozniak, M. Deep Neural Network Heuristic Hierarchization for Cooperative Intelligent Transportation Fleet Management. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 16752–16762. [CrossRef]
10. Zielonka, A.; Sikora, A.; Woźniak, M. Fuzzy rules intelligent car real-time diagnostic system. *Eng. Appl. Artif. Intell.* **2024**, *135*, 108648. [CrossRef]
11. Liu, X.; Gao, C.; Zhao, J.; Chen, Y. Deep Reinforcement Learning for Task Offloading in Vehicular Edge Computing. In Proceedings of the IEEE International Conference on Communications (ICC), Denver, CO, USA, 9–13 June 2024; pp. 1–6.
12. Moon, S.; Lim, Y. Federated Deep Reinforcement Learning Based Task Offloading with Power Control in Vehicular Edge Computing. *Sensors* **2022**, *22*, 9595. [CrossRef] [PubMed]
13. Ji, M.; Wu, Q.; Fan, P.; Cheng, N.; Chen, W.; Wang, J.; Letaief, K.B. Graph Neural Networks and Deep Reinforcement Learning-Based Resource Allocation for V2X Communications. *IEEE Internet Things J.* **2024**, *12*, 3613–3628. [CrossRef]
14. Polowczyk, A.; Polowczyk, A.; Woźniak, M. Graph Neural Network via Dynamic Weights and LSTM with Attention for Traffic Forecasting. *Procedia Comput. Sci.* **2025**, *257*, 119–126. [CrossRef]
15. Wu, J.; Zou, Y.; Zhang, X.; Liu, J.; Sun, W.; Du, G. Dependency-Aware Task Offloading Strategy via Heterogeneous Graph Neural Network and Deep Reinforcement Learning. *IEEE Internet Things J.* **2025**, *12*, 22915–22933. [CrossRef]
16. Busacca, F.; Palazzo, S.; Raftopoulos, R.; Schembra, G. MANTRA: An Edge-Computing Framework based on Multi-Armed Bandit for Latency- and Energy-aware Job Offloading in Vehicular Networks. In Proceedings of the 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), Madrid, Spain, 19–23 June 2023; pp. 143–151.
17. Chen, S.; Li, W.; Sun, J.; Pace, P.; He, L.; Fortino, G. An Efficient Collaborative Task Offloading Approach Based on Multi-Objective Algorithm in MEC-Assisted Vehicular Networks. *IEEE Trans. Veh. Technol.* **2025**, *74*, 11249–11263. [CrossRef]

18. *IEEE Std 802.11p-2010*; IEEE Standard for Information Technology—Telecommunications and Information Exchange between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Amendment 6: Wireless Access in Vehicular Environments. IEEE: New York, NY, USA, 2010.
19. Bose, S.K. *An Introduction to Queueing Systems*; Springer Science & Business Media: New York, NY, USA, 2013.
20. Brody, S.; Alon, U.; Yahav, E. How Attentive are Graph Attention Networks? *arXiv* **2022**, arXiv:2105.14491.
21. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In Proceedings of the 4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016.
22. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
23. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML), New York, NY, USA, 19–24 June 2016; pp. 1928–1937.
24. Zhao, Z.; Perazzone, J.; Verma, G.; Segarra, S. Congestion-Aware Distributed Task Offloading in Wireless Multi-Hop Networks Using Graph Neural Networks. In Proceedings of the ICASSP 2024—2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Seoul, Republic of Korea, 14–19 April 2024; pp. 8951–8955.