Journal of Computers Vol. 37 No. 5, October 2026, pp. 165-182
https://doi.org/10.63367/19911599202610370501 1

GDN-Based Dependency Task Offloading for Latency-Sensitive Vehicular MEC Networks

Ruijun Bai*

Gansu Computing Center, Lan Zhou, Gan Su, 730070, China
bairuij115@163.com

Received 26 July 2025; Revised 5 February 2026; Accepted 22 February 2026

Abstract. This paper investigates the critical challenge of dependent task offloading in vehicular edge computing, a scenario where computational tasks with complex interdependencies must be efficiently distributed across vehicular networks while simultaneously meeting stringent low-latency requirements and adapting to dynamic network conditions. Conventional task offloading approaches often fail to adequately balance the intricate topological relationships among interdependent tasks with the real-time state of heterogeneous edge resources, leading to suboptimal scheduling performance, increased latency, and inefficient resource utilization. These limitations are particularly pronounced in highly mobile environments characterized by rapidly changing network topologies and unpredictable vehicle movements. To address these challenges, we propose a novel Graph Neural Network-based Dependency-Task Offloading (GDTO) algorithm, which introduces a two-stage optimization framework. First, it intelligently prioritizes subtasks by comprehensively analyzing both task-specific attributes and system information. Second, it leverages a hybrid Graph Convolutional Network-Deep Q-Network (GCN-DQN) deep reinforcement learning framework to dynamically model task dependencies as graph structures while enabling adaptive decision-making in real time. This innovative integration allows GDTO to efficiently coordinate offloading decisions across multiple vehicles and roadside units (RSUs), accounting for spatial-temporal constraints and resource heterogeneity. Finally, comparative simulation experiments with four comparative algorithms demonstrate that the GDTO algorithm can effectively reduce the total task offloading and scheduling latency across various parameter environments.

Keywords: GCN, DAG, task offloading, RSU

1 Introduction

With the rapid development of Intelligent Transportation Systems (ITS) and Internet of Vehicles (IoV) technologies, modern vehicles are progressively evolving into mobile nodes with advanced sensing and computing capabilities [1-3]. These vehicles not only require real-time environmental data collection through onboard sensors to support critical functions such as autonomous driving and collision avoidance, but also need to maintain efficient communication with other vehicles, Roadside Units (RSUs), and cloud platforms to ensure reliable applications like real-time navigation and collaborative decision-making [4, 5]. However, as autonomous driving levels advance and intelligent traffic management demands grow, vehicular tasks are becoming increasingly complex and computationally intensive. Limited by constrained onboard computing resources, vehicles struggle to independently process all tasks, particularly those with stringent real-time requirements in highly dynamic traffic scenarios. Traditional cloud computing paradigms often fail to meet the real-time demands of IoV due to issues such as high network latency and bandwidth fluctuations. To address these challenges, Multi-access Edge Computing (MEC) has emerged as a promising solution by deploying computing resources at the network edge (e.g., near base stations or RSUs), providing vehicles with low-latency, high-reliability computing services [6-8]. MEC enables efficient processing of offloaded vehicular tasks, significantly optimizing system response times and enhancing user experience. Nevertheless, efficiently offloading and scheduling tasks - especially those with complex dependencies - in the dynamic IoV environment remains a critical unresolved challenge.

In practical vehicular network environments, computational tasks generated by vehicles are typically not isolated but exhibit inter-dependencies [9, 10]. A representative example emerges in autonomous driving

* Corresponding Author

scenarios, where vehicles must simultaneously process multiple sensor data streams with inherent sequential dependencies among these processing tasks. When the output of one task serves as the input for another, these tasks cannot be executed in parallel but must follow strict dependency constraints. Such interdependent tasks are formally modeled as Directed Acyclic Graphs (DAGs). Compared to conventional computational tasks, these dependency relationships significantly complicate the computation offloading problem. Traditional heuristic algorithms or static scheduling approaches often prove inadequate in adapting to dynamic network conditions and heterogeneous computing resources, potentially leading to increased task execution latency or suboptimal resource utilization. Consequently, the design of efficient dependency-aware task offloading algorithms that optimize both overall task completion time and energy consumption has become a critical research focus in the field.

In recent years, deep reinforcement learning (DRL) has been widely adopted for task offloading problems due to its exceptional environmental perception and decision-making capabilities. Among various DRL approaches, Graph Convolutional Networks (GCNs) have demonstrated remarkable effectiveness in capturing topological dependencies among tasks. By aggregating node features and neighborhood information, GCNs provide a global optimization perspective for offloading dependent tasks. For instance, when integrated with attention mechanisms, GCNs can dynamically adjust task priorities to optimize MEC resource allocation. Another prevalent approach is the Deep Q-Network (DQN), which learns optimal offloading policies through Q-value iteration, making it particularly suitable for high-dimensional state space decision-making. Enhanced variants of DQN, including Double DQN and Dueling DQN, have further improved algorithmic stability and convergence speed. However, current research predominantly focuses on static or single-MEC scenarios, failing to adequately address the dynamic challenges posed by vehicle mobility and multi-MEC collaboration.

In practical vehicular network environments, the dynamic mobility of vehicles presents substantial challenges for task offloading decisions [11]. The high-velocity movement of vehicles induces continuous fluctuations in network topology and wireless channel conditions, which directly deteriorates the communication quality between vehicles and edge servers, consequently impairing task execution efficiency. The cooperative scheduling among multiple MEC servers further exacerbates the system complexity. First, the dynamically changing connectivity between vehicles and MEC servers may cause task interruptions or migrations, introducing additional computational and communication overhead. Second, dependent tasks require careful distribution across multiple MEC servers to simultaneously achieve load balancing and minimize cross-server communication latency [12]. Particularly challenging is the exponential growth of the solution space for offloading strategies as the number of MEC servers increases, owing to the interdependencies among tasks where the offloading decision for one subtask critically influences the execution locations of other related tasks. This combinatorial explosion significantly amplifies the optimization complexity.

Despite significant theoretical and practical advancements in computation offloading techniques, existing research has paid insufficient attention to dependent task offloading. Current approaches predominantly focus on dependent tasks from single mobile devices while neglecting the critical challenges of concurrent multi-vehicle offloading across multiple MEC servers. Moreover, these studies typically assume static scenarios where vehicles remain stationary and no competing tasks contend for edge server resources, assumptions that ignore vehicle mobility and substantially deviate from real-world conditions.

To address these limitations, this paper proposes a novel hybrid optimization framework integrating GCN and DRL. Our solution uniquely considers both vehicle mobility and concurrent offloading of dependent tasks from multiple vehicles. By modeling task dependencies as DAGs and leveraging graph neural networks to capture intricate inter-task relationships, the proposed algorithm achieves optimal sub-task offloading decisions in dynamic vehicular environments.

The main contributions of this work are threefold:

- We propose a novel GCN-DQN(GDN) based framework that simultaneously models both task dependencies and dynamic network topology, enabling real-time capture of inter-task correlations and inter-server relationships. This approach effectively addresses the challenge of characterizing complex dependencies in mobile environments.
- We develop an innovative GDN-based Dependency-Task Offloading (GDTO) algorithm that uniquely combines the structural awareness of GCNs with the decision-making capability of DQN. Our solution achieves distributed decision-making in multi-MEC environments by: (1) employing GCNs to extract task dependency patterns, and (2) utilizing DQN for optimal offloading policy generation, thereby optimizing long-term task completion rates.
- Through extensive simulations, we evaluate our method against four representative benchmarks: (1) DDPG-off algorithm, (2) PPO-off algorithm, (3) Greedy-off algorithm, and (4) Rand-Off algorithm. The

comprehensive results demonstrate that our method achieves minimal offloading latency across diverse parameter configurations, proving its superior performance.

The remainder of this paper is organized as follows: Section II reviews related work, including fundamental research on graph neural networks and computation offloading techniques. Section III elaborates on the system model and provides a rigorous problem formulation. Section IV presents in detail our proposed GCN-DQN optimization algorithm for dependent task offloading. Section V provides comprehensive simulation results and performance analysis. Finally, Section VI concludes the paper and discusses potential directions for future research.

2 Related Work

In the field of edge computing, the task offloading problem has received significant attention, particularly in the context of the IoV. Numerous solutions have been proposed to address this challenge [13-17]. Wu et al. [18] proposed a DRL-based hybrid task offloading scheme that simultaneously considers latency constraints and resource requirements in the IoV task offloading problem, which can reduce task latency and energy consumption while achieving efficient resource management. Wang et al. [19] investigated the dependent task offloading problem under different network conditions in MEC-based environments and proposed a DQN-based algorithm. Zou et al. [20] presented an online task scheduling architecture based on the Proximal Policy Optimization (PPO) algorithm to optimize the offloading latency of dependent tasks. Lou et al. [21] considered the dependent task offloading problem under resource and bandwidth constraints, proposing to generate cloud clones of tasks to fully utilize cloud resources, thereby releasing the bandwidth and computational resources consumed by redundant tasks. Zhang et al. [22] studied a MEC-supported task offloading architecture for vehicular networks, proposing a task offloading scheme based on task classification and offloading node selection, with simulation experiments verifying that the scheme can indeed reduce the total latency of task execution under latency constraints in vehicular networks. Aslam et al. [23] investigated the joint optimization problem of task offloading latency and handover cost in fog computing systems for vehicular networks, developing an efficient parallel offloading scheme. While these studies demonstrate significant progress in IoV task offloading, they primarily focus on isolated performance metrics (e.g., latency, energy, or bandwidth) and often overlook the complex interdependencies in real-world vehicular networks, such as high mobility, unstable connectivity, and multi-layer resource constraints. This limitation motivates the need for a more comprehensive approach that jointly optimizes offloading decisions across dynamic edge environments.

The dependent task offloading problem in vehicular networks is a major challenge in task offloading research [24]. Al-Habob et al. [25] proposed a conflict-free sequential task offloading scheme for dependent tasks and verified through simulations that their approach effectively reduces latency and offloading failure probability. [26] investigated the DAG-based task offloading problem in a cloud-edge-end collaborative framework, aiming to minimize application completion time. He et al. [27] studied the problem of multiple tasks offloaded to the same node in UAV-assisted communication networks and introduced a novel scheduling algorithm. Cui et al. [28] explored a distributed vehicle-to-vehicle (V2V) task offloading scheme, where vehicles with computing capabilities serve as computing nodes to cooperatively assist task vehicles in resolving offloading challenges. Despite these advances, the inherent complexity of dependent task offloading means no existing algorithm can comprehensively address all dependency challenges, leaving significant room for further research development.

GCNs can effectively utilize the correlation information between dependent tasks to achieve efficient offloading, making them frequently adopted in dependent task offloading problems. Mo et al. [29] proposed a novel GCN and DRL-based algorithm that transforms the dependent task offloading problem into a node classification problem using GCN, with simulation results verifying the model's ability to rapidly converge across various DAG structures. Chen et al. [30] considered the dynamic changes in network environments and developed an algorithm combining GCN with Deep Deterministic Policy Gradient (DDPG) to minimize both task offloading time and energy consumption. Chen et al. [31] investigated dependent task offloading in multi-user scenarios, comprehensively considering task topology and channel interference, and proposed an Actor-Critic network-based offloading algorithm. Ye et al. [32] studied dependent task offloading in multi-user, multi-MEC server environments with the objective of minimizing task completion time, proposing a deep DRL framework called GDDTO that employs GCN to extract task dependencies while integrating Double Deep Q-Network (DDQN) and an optimized experience replay mechanism to select and evaluate offloading strategies. Yang et al. [33] presented an RL-based container scheduling algorithm incorporating GCN technology to handle

dependent task offloading, with extensive simulations validating the algorithm's superiority. While GCN-based approaches show promising results in handling task dependencies, their performance remains constrained by computational complexity and generalization challenges across diverse network environments, suggesting need for more adaptive solutions.

While most existing studies (Table 1) address only a single or a subset of dimensions—task priority scheduling, environmental dynamics, and multi-MEC collaboration—the proposed GDTO algorithm addresses the limitations by integrating a GCN and DQN architecture. It achieves a comprehensive improvement in dependent task offloading efficiency for complex vehicular networks through the co-optimization of subtask priority ordering, dynamic offloading policy adaptation, and multi-device offloading scheduling mechanisms.

Table 1. Comparison of related work

Solutions	Task priority scheduling	Environmental dynamics	Multi-MEC collaboration
[19, 21]	✓	×	×
[20, 23]	✓	✓	×
[25, 26, 29, 33]	✓	×	✓
[22, 28, 30-32]	×	✓	✓
This work	✓	✓	✓

3 System Model

3.1 Channel Modelling

In vehicular networks, since the computational resources of vehicles themselves are limited, computational tasks are often offloaded to adjacent wireless edge nodes to reduce task processing time. As shown in Fig. 1, multiple RSUs equipped with servers possessing computational resources are deployed along the roadside. Each RSU is responsible for handling the task offloading and computation for vehicles within a specific road segment. Each vehicle can choose to either compute tasks locally or upload them to the nearest RSU for processing. The RSU then returns the computed results back to the vehicle upon task completion. As vehicles move continuously, they must dynamically switch their optimal offloading node selection to minimize task scheduling latency during the offloading process. We consider a vehicular network consisting of one RSU and multiple task vehicles within its coverage area, where the set of task vehicles is denoted as $V = \{1, 2, \ldots, V\}$, and the set of RSUs equipped with edge servers is denoted as $M = \{1, 2, \ldots, M\}$.

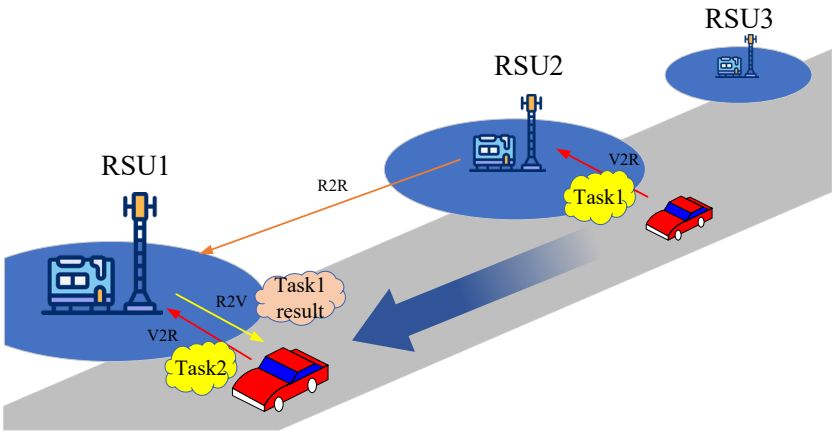


Fig. 1. IoV task offloading model diagram

We regard all computational tasks generated by each task vehicle within a certain period as a single integrated task. Typically, each computational task generated by a vehicle can be decomposed into multiple subtasks, which have dependencies among them. These dependencies can be represented by a Directed Acyclic Graph (DAG), as shown in Fig. 2, where vertices represent subtasks and edges represent dependencies between them. Each subtask is denoted by a binary tuple: $n_{i,j} \triangleq \{D_{i,j}, C_{i,j}\}$, where $n_{i,j}$ represents the j-th subtask of the i-th vehicle, $D_{i,j}$ denotes the data volume of this subtask (which affects the transmission time), and $C_{i,j}$ represents the computational workload of the subtask, expressed in terms of the required CPU cycles. For clarity, we define the set of predecessor subtasks for each subtask as $pred_{i,j}$, and the set of successor subtasks as $succ_{i,j}$. A subtask cannot begin execution until all its predecessor subtasks are completed. Similarly, successor subtasks can only start execution after the current subtask is finished.

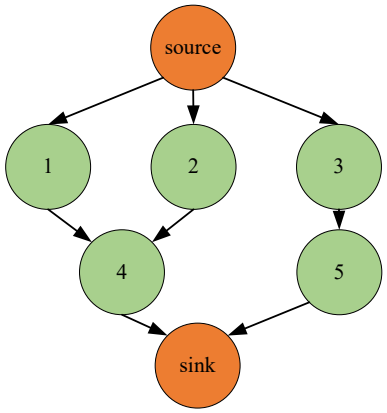


Fig. 2. Task dependency DAG diagram

If the subtask is offloaded to RSU m, the data transmission delay between the vehicle and the RSU must be considered. At system time t, the V2I (Vehicle-to-Infrastructure) link transmission rate between vehicle i and RSU m is given by:

$$R_{i,m}(t) = B\log_2\left(1+\frac{P_i h_{i,m}(l_{i,m}(t))^{-\alpha}}{\sum_{i'\in V, i'\neq i} P_{i'} h_{i',m}(l_{i',m}(t))^{-\alpha}+\sigma^2}\right), \tag{1}$$

where B represents the subchannel bandwidth, P_i denotes the transmission rate of vehicle i, $h_{i,m}$ indicates the channel gain between vehicle i and the RSU m when offloading tasks (the channel gain follows a Rayleigh distribution model), $l_{i,m}$ represents the distance between the vehicle and the RSU, and σ^2 stands for the Gaussian white noise power.

Typically, the execution result data of a task is very small and negligible compared to the size of the task itself. Therefore, we only consider the uplink transmission delay $t^{tr}_{i,j}$, which is calculated as:

$$tr^m_{i,j} = \frac{D_{i,j}}{R_{i,m}}. \tag{2}$$

When a subtask is offloaded to the RSU m, it gets executed by the RSU's computing resources. Then the computation time of subtask $n_{i,j}$ can be denoted as $tc^m_{i,j}$, which is formulated as:

$$tc_{i,j}^{m} = \frac{C_{i,j}}{f_m}, \tag{3}$$

where f_m denotes the local processing capacity of RSU m, specifically its CPU frequency measured in cycles per second (cycles/s).

If local computation is selected, the required task data transmission time is $tc_{i,j}^{0} = 0$. Let f_0 represent the computational capability of the vehicle itself, the local execution time $tc_{i,j}^{0}$ of subtask j on vehicle i can be expressed as:

$$tc_{i,j}^{0} = \frac{C_{i,j}}{f_0}. \tag{4}$$

The start time ST of a subtask is constrained by two factors: (1) the earliest available time RT of the computing device, and (2) the maximum completion time FT of its predecessor tasks. On one hand, due to limited computing power and memory resources, we assume each computing device can only process one subtask at a time under a First-Come-First-Served (FCFS) policy. Thus, the subtask must wait until all queued tasks on the device are completed before execution can begin, meaning its start time must be later than the device's available time. On the other hand, the subtask is subject to dependency constraints and can only start computation after all subtasks in predecessor task $pred_{i,j}$ have finished. Therefore, the start time ST can be expressed as:

$$ST_{i,j}(A_{i,j} = m) = \max\{RT_m, \max_{k \in pred_{i,j}} \{FT_{i,k}\}\}, \tag{5}$$

where RT_m denote the earliest available time of the RSU server m, which specifically represents the completion time of the most recent subtask offloaded to this computing device. Upon scheduling completion of each subtask, RT_m is updated according to the completion time value $FT_{i,j}$ of that subtask $n_{i,j}$.

$$RT_m(t+1) = RT_m(t) + FT_{i,j} - ST_{i,j}. \tag{6}$$

When the task vehicle selects local computation $A_{i,j} = 0$, the task completion time FT is solely determined by the computation time $tc_{i,j}^{0}$. If the task vehicle chooses RSU offloading $A_{i,j} = m$, $m \neq 0$, then the task completion time FT depends on both the data transmission time $tr_{i,j}^{m}$ and computation time $tc_{i,j}^{m}$. Therefore, the completion time of subtask offloading computation can be uniformly expressed as:

$$FT_{i,j} = \begin{cases} tc_{i,j}^{0}, & \text{if } A_{i,j} = 0, \\ tr_{i,j}^{m} + tc_{i,j}^{m}, & \text{otherwise.} \end{cases} \tag{7}$$

The total latency T of a task can be represented by the completion time $FT_{i,j}$ of the last-finishing subtask, i.e.,

$$T_i = \max\{FT_{i,j}\}. \tag{8}$$

We aim to optimize the total task offloading latency, with the objective defined as minimizing the cumulative task processing latency for all vehicles within the RSU's coverage zone. Thus, the dependency-aware task offloading latency minimization problem can be formally formulated as:

$$
\begin{aligned}
&\text{P: } \min \sum_{i \in \mathcal{V}} T_i \\
&\text{s.t. } R_{i,m}(t) = B \log_2 (1 + \frac{P_i h_{i,m}(l_{i,m}(t))^{-\alpha}}{\sum_{i' \in \mathcal{V}, i' \neq i} P_{i'} h_{i',m}(l_{i',m}(t))^{-\alpha} + \sigma^2}), \\
&\quad tr_{i,j}^m = \frac{D_{i,j}}{R_{i,m}}, \\
&\quad tc_{i,j}^m = \frac{C_{i,j}}{f_m}, \\
&\quad tc_{i,j}^0 = \frac{C_{i,j}}{f_0}, \\
&\quad ST_{i,j}(A_{i,j} = m) = \max\left\{RT_m, \max_{k \in \text{pred}_{i,j}} \{FT_{i,k}\}\right\}, \\
&\quad RT_m(t+1) \ = \ RT_m(t) \ + \ FT_{i,j} \ - \ ST_{i,j}, \\
&\quad FT_{i,j} = \begin{cases} tc_{i,j}^0, & \text{if } A_{i,j} = 0, \\ tr_{i,j}^m + tc_{i,j}^m, & \text{otherwise}, \end{cases} \\
&\quad T_i = \max\{FT_{i,j}\}, \\
&\quad i \in \mathcal{V}, j \in \mathcal{N}, A \in \{0,1\}.
\end{aligned}
\tag{9}
$$

4 GDN-based Dependency-Task Offloading Algorithm

In this section, we first model the problem as an Markov Decision Process (MDP) model, then design a dependency-aware task offloading algorithm based on the combination of GCN and DQN, outline the architecture of this algorithm, and explain how it interacts with the MEC system.

4.1 Modeling the Problem as MDP

Since tasks have dependencies that mutually influence each other, if the state is designed to only consider the current subtask, it would struggle to handle long-range or cross-task dependencies. On the other hand, incorporating all subtask information would introduce parameter redundancy and cause the state dimension to grow linearly with the number of subtasks, potentially leading to the curse of dimensionality. The GCN framework can effectively capture connectivity between nodes. Through convolutional operations, it aggregates features from each node and its neighbors, generating node embeddings that incorporate local structural information. This makes it particularly suitable for handling tasks with complex interdependencies.

Since task offloading decisions depend only on the previous state, we model all dependency tasks generated by vehicles as a MDP. An MDP can be represented by a tuple $<S_t, A_t, R_t, S_{t+1}>$, where each element is defined as follows:

- **State** S**:** We consolidate all information affecting vehicular computation offloading latency into the system state. Each subtask is treated as a node in the graph, thus the state incorporates both subtask feature information and dependency relationships between subtasks. Furthermore, since the computational capability and idle time of MEC servers significantly impact offloading decisions, we model MEC servers as special nodes in the graph, with their relevant information included in the state. To maintain consistency with the total node count, we represent both inter-subtask dependencies and subtask-server offloading relationships as an adjacency matrix $G \in \mathbb{R}^{(N+M)\times(N+M)}$, where each element is denoted by $g_{i,j}$. The matrix is formally expressed as:

$$g_{i,j} = \begin{cases} 1, \text{ if } i, j < N,\ i \in pred_j,\ j \in succ_i, \\ 0, \text{ if } i, j < N,\ i \notin pred_j,\ j \notin succ_i, \\ 1, \text{ if } i < N,\ j > N,\ A_i = j, \\ 0, \text{ otherwise} \end{cases} \tag{10}$$

In summary, state S contains the following information: the one-hot encoded subtask sequence E, subtask data volume information D, subtask computational load information C, adjacency matrix G, server computational capability information f, and server idle time information RT. The formula is expressed as:

$$S = \left\{ s_i \middle|\ s_i = (E, D, C, f, RT, G) \right\}, \quad 1 \le i \le N \tag{11}$$

where RT is initialized to 0, and the state S is updated each time a subtask is offloaded.

- **Action A:** In each iteration, one subtask is offloaded, and each subtask can be assigned to any available server. Thus,

$$A_{i,j} = \{ m \mid\ m \in M \} \tag{12}$$

where $m = 0$ represents local computation, and $m > 0$ represents offloading to the RSU m for computation and processing.

- **Reward Function r:** Since the objective is to minimize task offloading time, to evaluate the impact of each offloading action on the final completion time, we set the reward function as the negative value of the total time increment after executing the current offloading action. That is:

$$r = \max(FT_{i,j-1}) - \max(FT_{i,j}) \tag{13}$$

where $\max(FT_{i,j-1})$ represents the maximum completion time among the first $j-1$ subtasks, and $\max(FT_{i,j})$ denotes the maximum completion time after offloading this subtask $n_{i,j}$. The smaller the increase in the maximum completion time, the higher the reward value.

4.2 Subtask Priority Sorting

For a DAG task, there are multiple execution paths from the source node to the sink node, as shown in Fig. 3. The choice of different paths directly affects the total scheduling latency of the system. According to the critical path theory, the total scheduling latency of task offloading is determined by the longest path (i.e., the critical path). Therefore, when allocating computing resources, priority should be given to the subtasks on the critical path. Based on this theory, we propose a subtask priority scheduling algorithm that first ranks all subtasks by priority and then allocates computing devices to them sequentially in order of priority.

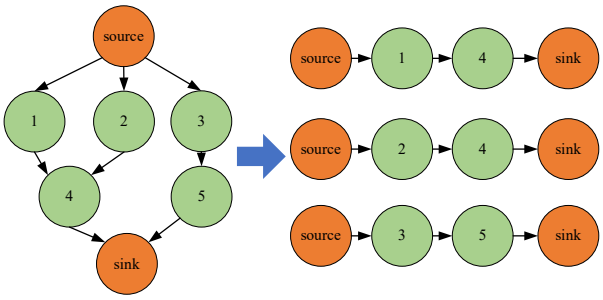


Fig. 3. Parallel execution paths in task dependencies

To estimate the offloading and scheduling time for each subtask, it is first necessary to calculate the average computational power and average transmission rate of all offloadable RSUs.

$$\overline{f}_m = \frac{\text{sum}(f_m)}{M}, \tag{14}$$

$$\overline{R}_{i,m} = \frac{\text{sum}(R_{i,m})}{M}. \tag{15}$$

Subsequently, the estimated computation time $\overline{tc}_{i,j}$ and transmission time $\overline{tr}_{i,j}$ for each subtask are calculated, expressed by the following formulas:

$$\overline{tc}_{i,j} = \frac{C_{i,j}}{\overline{f}_m}, \tag{16}$$

$$\overline{tr}_{i,j} = \frac{D_{i,j}}{\overline{R}_{i,m}}. \tag{17}$$

Finally, the priority $prio_{i,j}$ of each subtask is iteratively computed using the following formula:

$$prio_{i,j} = \max_{k \in succ_{i,j}} \{prio_{i,k}\} + \overline{tc}_{i,j} + \overline{tr}_{i,j}. \tag{18}$$

The subtasks are sorted according to their priorities $prio_{i,j}$, and the resulting ordered sequence is denoted as Φ.

4.3 GDN Framework

In this subsection, we propose a GDN-based Dependency-Task Offloading (GDTO) algorithm for IoV, whose framework is illustrated in Fig. 4.

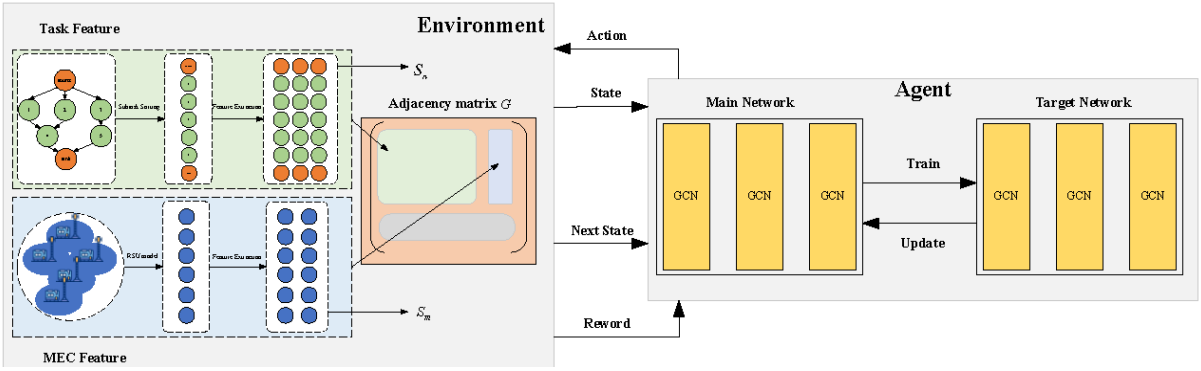


Fig. 4. GDN-based Dependency-Task Offloading algorithm framework

In the GDTO framework, the critical path method is first employed to compute the priority sequence $prio_{i,j}$ of subtasks. Subsequently, the feature information of all subtasks is extracted as state S_n, while the characteristic parameters of each RSU are captured as state S_m. The dependency relationships among subtasks and the

offloading associations between subtasks and RSUs are jointly represented by an adjacency matrix G. In G, the first n rows and n columns encode the interdependencies among subtasks, whereas the entries from column n+1 to column n+m in the first n rows denote the offloading relations between subtasks and RSUs. Together, S_n, S_m, and G constitute the system environment state. These variables are fed into the main network within the intelligent agent, where a GCN processes them to generate offloading decisions. The resulting decisions are then embedded into G, forming the updated environmental variables for the next iteration. This cyclic process continues until an offloading decision that satisfies the loss function requirements is obtained.

DQN evaluates each offloading decision by estimating its expected long-term return through a value function, which subsequently guides the adjustment of the training network parameters. As the training network iteratively refines its parameters, the optimized parameters are transferred back to the main network for updating, thereby generating improved offloading decisions. This iterative loop continues until the network parameters converge.

This framework holistically integrates task-related information, inter-subtask dependencies, RSU characteristics, and spatial factors such as distances between RSUs and task vehicles into a unified environmental representation. By synergistically combining the structural awareness of GCN with the sequential decision-making capability of DQN, the proposed approach effectively reduces the overall task offloading and scheduling duration while significantly enhancing the utilization efficiency of system computing resources.

For problem P, the challenge lies in how to handle the dependencies between subtasks and make effective offloading decisions. To address this, we adopt an algorithm that combines GCN and DQN to efficiently learn the offloading strategy. RL can be broadly categorized into two types: value-based reinforcement learning and policy-based reinforcement learning. The former learns a value function and then derives a policy based on it, without explicitly maintaining a policy during the learning process. Specifically, it uses the action-value function to represent the expected future reward of an action in a given state, which can be calculated as follows:

$$Q(s_t, a_t) = \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots], \tag{19}$$

where π represents the current execution policy, and γ denotes the discount factor for future rewards. The goal is to find the optimal policy $Q(s_t, a_t)$ that maximizes the cumulative reward.

The DQN algorithm is a value-based reinforcement learning method that employs a neural network to approximate a nonlinear function, mapping input states to the corresponding $Q(s_t, a_t)$ values for each action. Since the problem involves multiple subtasks, the dependencies between them can be represented by an adjacency matrix. To handle this, we integrate a GCN into the DQN framework to process subtask data and learn the offloading strategy. We utilize two neural networks for joint training: a training network $\pi(\theta)$ and a target network $\pi(\theta^*)$, where θ denotes the weight parameters of the networks. The training network is responsible for learning the offloading policy and generating actions, with its parameters updated based on reward feedback from the environment. To avoid local optima, we employ the $\varepsilon-$ greed method to balance exploitation and exploration. The process can be formally expressed as:

$$a_t = \begin{cases} \arg\max Q(s_t, a_t \mid \theta), \text{ with probability } (1-\varepsilon), \\ \text{Random select an action, } \quad \text{with probability } \varepsilon. \end{cases} \tag{20}$$

After each round of action execution, the data (s_t, a_t, r_t, s_{t+1}) is stored in the experience replay buffer. The target network uses the cached training data to compute the loss value and updates its weights after a certain number of iterations. By uniformly sampling a batch of data from the experience replay buffer, the loss function can be expressed as:

$$L(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1})}\left[\left(r_t + \gamma \max Q(s_{t+1}, a_{t+1} \mid \theta^*) - Q(s, a \mid \theta)\right)^2\right]. \tag{21}$$

The training network updates its parameters via gradient descent on the loss function, while the target network updates its weights by copying parameters from the training network every K steps.

During training, we randomly generate multiple DAG tasks following an identical distribution and store them in set J. At the start of each episode, a DAG task is randomly selected from J. At each step, a subtask is offloaded by selecting an action through the actor network, and a corresponding reward is obtained from the environment.

The detailed algorithmic procedure is presented in Algorithm 1.

Algorithm 1. GDTO algorithm

```
for episode in max episode do
    Expand task dependency relationships and offloading matrix into adjacency matrix
    Convert task features E, D, C and RSU feature RT into linear vectors
    Concatenate the converted E, D, C, RT, and G to obtain the initial state s_0
    for t in N do
        Randomly select an action A_t based on probability ε, otherwise choose action A_t
based on maximizing the Q function,
        Calculate the current reward r_t
        Update the unloading matrix Y and adjacency matrix G based on actions
        Confirm the next subtask n_j based on priority of the task
        Update the next state s_{t+1}
        Store (S_t, A_t, r_t, s_{t+1}) in the experience replay buffer
        Randomly sample small batches of data from the experience replay buffer
        Calculate Q* based on the extracted data
        Calculate the loss L based on Q* and Q
        Update the GDN network parameter θ using gradient descent and learning rate l_r
to minimize the loss L
        Update max(FT_{i,j}) after uninstalling this subtask
        Gradually reducing the randomness of exploration ε
    end for
    Record the results of this episode max(FT_{i,j})
end for
```

The GDN model comprises three graph convolutional layers and two fully connected layers, decomposing the state into feature vectors and the adjacency matrix G in the first layer. The model training consists of two cycles. In the minor loop, each subtask is offloaded sequentially, determining the reward function and the next state while updating partial network parameters. Once all subtasks are offloaded, a new major loop begins, resetting the environment and offloading the entire task again, with all network parameters being updated during the first minor loop.

5 Simulation and Analysis

To validate the effectiveness of the proposed task offloading optimization algorithm for vehicular networks, this chapter establishes a simulation platform based on Python 3.9. Comparative experiments with benchmark schemes are conducted to evaluate system performance using total delay as the metric. Section 5.1 elaborates on parameter configurations and the design of comparative schemes, while Section 5.2 provides an in-depth analysis of the proposed algorithms optimization efficacy through multi-scenario variable-parameter tests.

5.1 Simulation Settings

The parameter settings for the simulation experiment are shown in Table 2.

Table 2. Simulation parameter settings

Description	Parameters
Number of RSUs	$M = 6 \sim 10$
Number of task vehicles	$V = 5$
Number of sub-tasks in DAG tasks	$N = 100 \sim 500$
Density of directed edges for DAG tasks	$0.2 \sim 0.6$
CPU frequency of RSU servers /GHz	$f_s = 7 \sim 12$
CPU frequency of vehicles /GHz	$f_v = 4$
Required CPU frequency of subtask /GHz	$C = 0.1 – 0.3$
Sub-task upload data size /MB	$D = 10 \sim 20$
System Bandwidth /MHz	$B = 20$
Vehicles transmit power /W	$p_v = 0.1$
Noise power /dBm	$\sigma^2 = -74$

To evaluate the algorithm's performance, we compare our proposed method with the following approaches:

1. **PPO-Off (Proximal Policy Optimization-based Offloading) Algorithm:** A policy gradient algorithm that optimizes offloading decisions through proximal policy optimization. It continuously updates its stochastic policy to maximize long-term rewards while maintaining training stability through clipped objective updates.
2. **DDPG-Off (Deep Deterministic Policy Gradient-based Offloading) Algorithm:** An actor-critic method that combines DQN with deterministic policy gradients. It outputs continuous-valued actions, making it suitable for solving continuous action space problems.
3. **Greedy-Off (Greedy Offloading) Algorithm:** Always selects the currently fastest-available computing node (either RSU or onboard server) based on immediate idle time, without considering future task dependencies or resource contention.
4. **Rand-Off (Random Offloading) Algorithm:** Makes completely random offloading decisions between RSUs and local vehicle servers, serving as a performance baseline.

5.2 Result Analysis

Fig. 5 demonstrates the convergence performance comparison among the proposed GDTO algorithm and two baseline methods (PPO-Off and DDPG-Off). The horizontal axis represents the number of training iterations (episodes), while the vertical axis indicates the total time consumption for task offloading and computation. All three algorithms exhibit rapid convergence during the initial phase, with the total execution time decreasing significantly in the first 100 iterations. The curve trend demonstrates that the GDTO algorithm achieves rapid convergence during the initial phase, with the total execution time decreasing significantly as iterations progress. This improvement primarily stems from two factors: (1) the policy network begins stable parameter updates after accumulating sufficient training samples, and (2) the exploration probability ε gradually decays during training, resulting in more efficient agent decision-making. After approximately 300 iterations, the total time consumption stabilizes with markedly reduced fluctuations, indicating that the algorithm has approached convergence.

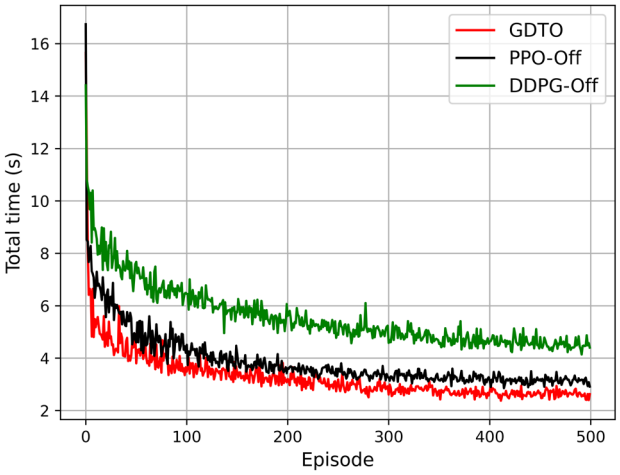


Fig. 5. Convergence curve of network model

Fig. 6 compares the task offloading time performance of five algorithms under varying subtask quantities (ranging from 100 to 500). The horizontal axis represents the number of subtasks, while the vertical axis indicates the total offloading time. As demonstrated, the proposed algorithm consistently outperforms all baseline methods, achieving the shortest processing time across all scenarios. In contrast, the random algorithm exhibits the poorest performance with the longest duration. Notably, the performance gap between algorithms widens as subtask count increases - while all algorithms show degraded performance with larger subtask quantities, the proposed method demonstrates superior scalability. This phenomenon occurs because increased subtask numbers

create greater optimization potential and amplify the importance of task prioritization, allowing our algorithm to better exploit its advanced scheduling mechanisms.

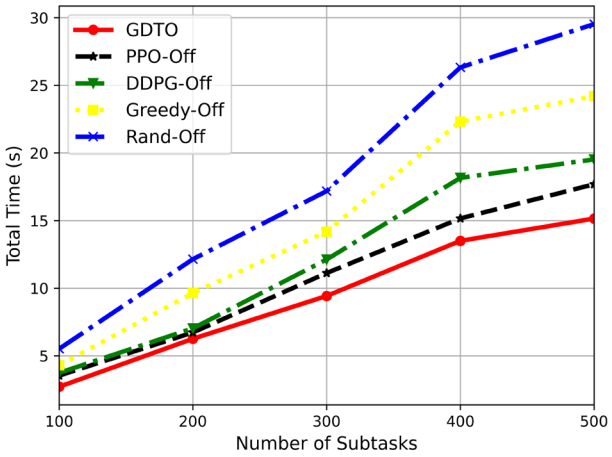


Fig. 6. Algorithm comparison of task offloading time under different subtask counts

Fig. 7 compares the performance of five task offloading algorithms under varying DAG edge densities, where edge density quantifies the probability of dependency relationships between subtasks. The results reveal a consistent upward trend in offloading time across all algorithms as edge density increases, which can be attributed to the growing number of precedence constraints that force subsequent tasks to wait for their predecessors to complete before execution. However, the rate of time increase exhibits a diminishing pattern at higher edge densities. This nonlinear behavior stems from two fundamental characteristics of DAG-based scheduling: first, the critical path length (determined by the longest chain of dependent tasks) becomes the dominant factor rather than the total number of dependencies, meaning additional edges beyond a certain point may not extend the overall schedule; second, as the graph approaches full connectivity, the dependency structure converges toward a near-sequential execution pattern where most tasks have single predecessors, thereby reducing the compounding effect of multiple dependencies. Notably, the proposed algorithm demonstrates remarkable robustness to these topological variations by employing an adaptive scheduling policy that dynamically prioritizes tasks based on both their dependency depth and resource requirements, particularly excelling in high-density scenarios where conventional methods struggle with the combinatorial complexity of dependency resolution.

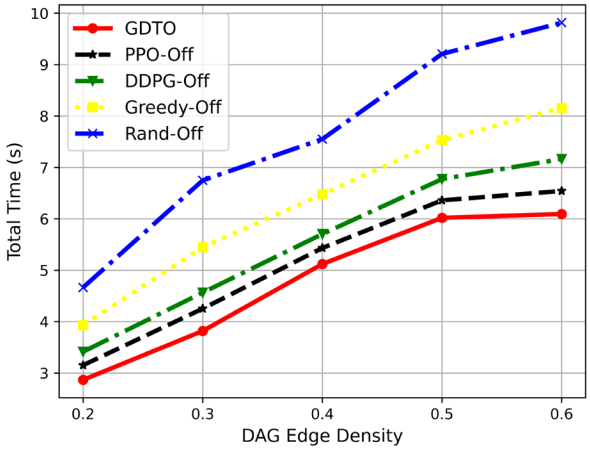


Fig. 7. Algorithm comparison of task offloading time under different DAG edge densities

Fig. 8 compares the task offloading time of five algorithms under varying numbers of RSUs, with fixed parameters of 100 subtasks and 0.2 DAG edge density. The experimental results reveal that while the task execution time decreases with additional RSUs, the marginal improvement diminishes progressively. This nonlinear scaling pattern stems from two distinct operational regimes: in the initial phase, increased RSU availability enhances scheduling flexibility and parallel processing capacity, enabling concurrent offloading of more subtasks. However, beyond a critical RSU threshold, the performance becomes fundamentally constrained by task dependencies - even with available RSU resources, subtasks remain blocked awaiting completion of their DAG-constrained predecessors. Most significantly, the performance gap between algorithms amplifies substantially with more RSUs, demonstrating that our proposed algorithm achieves superior scalability through its advanced two-tier optimization: (1) intelligent task-to-RSU mapping that minimizes communication overhead, and (2) dependency-aware load balancing that dynamically adjusts to real-time resource availability while respecting precedence constraints. These mechanisms become increasingly impactful in resource-abundant environments, where conventional algorithms fail to fully exploit the available parallelism due to their static scheduling approaches.

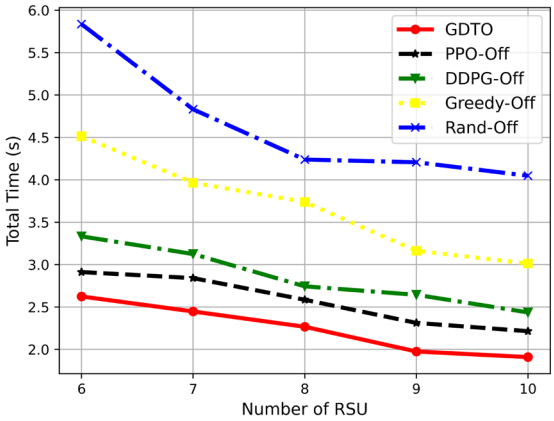


Fig. 8. Algorithm comparison of task offloading time under different RSU counts

Fig. 9 demonstrates the computational time trends of different algorithms under varying numbers of subtasks. The total computation time of subtasks constitutes the most significant portion of the overall offloading scheduling time, making the optimization of computational time the primary focus for improving total offloading duration and the key consideration in offloading device selection. As the number of subtasks increases, all algorithms exhibit an upward trend in total computation time, with GDTO consistently maintaining the lowest computational latency. This advantage primarily stems from GDTO's ability to accurately identify dependency features among tasks through GCN and dynamically optimize offloading decisions using DRL, thereby selecting the most suitable offloading device for each subtask in real-time. In contrast, PPO-Off and DDPG-Off algorithms show significant performance degradation with larger numbers of subtasks due to their inadequate modeling of task dependencies, while Greedy-Off and Rand-Off algorithms demonstrate substantially higher computation times owing to their lack of global optimization capabilities. The experimental results confirm that by effectively optimizing the critical factor of computation time, GDTO significantly enhances overall offloading scheduling efficiency.

Fig. 10 presents a scatter plot comparing per-subtask offloading and computation times of the three algorithms (GDTO, PPO-Off, and DDPG-Off) under 500 subtasks. The x-axis represents the start time of each subtask, while the y-axis indicates the time consumption for offloading and computation per subtask. As shown in the figure, GDTO's data points are clustered in the lower-left region, demonstrating consistently lower and more stable time consumption throughout the entire subtask offloading process. In contrast, PPO-Off and DDPG-Off exhibit significantly greater volatility, with occasional time spikes exceeding 0.2 seconds. These results clearly illustrate GDTO's superior stability and efficiency in handling sequential subtask execution compared to the baseline approaches.

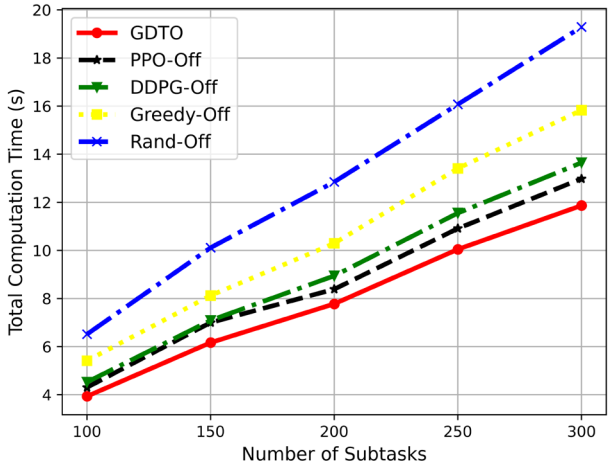
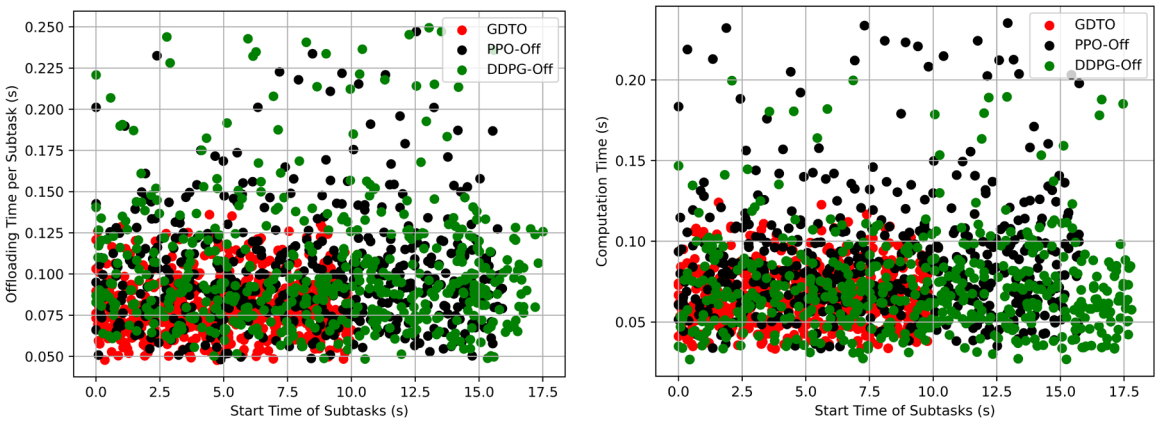


Fig. 9. Algorithm comparison of total computation time under different subtask numbers



(a) Offloading time per subtask versus Start time of subtasks (b) Computation time per subtask versus Start time of subtasks

Fig. 10. Algorithm comparison of Per-Subtask offloading and Computation time

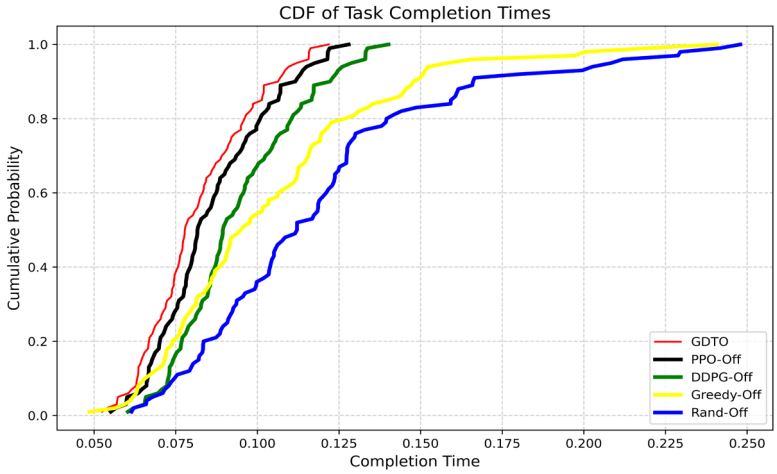


Fig. 11. Algorithm comparison of subtask offloading time CDF under 100 subtasks

Fig. 11 presents the cumulative distribution function (CDF) of task completion times for five offloading algorithms. The CDF curves demonstrate that GDTO achieves the steepest ascent, indicating its superior ability to complete tasks within shorter time delays. Specifically, over 80% of tasks are completed within 0.10s under GDTO, while the baseline algorithms (PPO-Off and DDPG-Off) require 0.125s to achieve the same completion rate. The greedy and random strategies exhibit the poorest performance, with only 60-70% of tasks finished by 0.250s. These results quantitatively validate GDTO's effectiveness in minimizing tail latency and improving overall task completion efficiency compared to conventional approaches.

6 Conclusion and Future Work

This paper systematically investigates the optimization problem of dependent task offloading in vehicular edge computing and proposes a GDN-based Dependency-Task Offloading (GDTO) algorithm integrating GCN and DQN. The main contributions and innovations of this study are as follows:

1. **A hybrid GCN-DQN architecture (GDN) that integrates graph structure and reinforcement decision-making is constructed.** This architecture achieves, for the first time, the organic fusion of graph convolutional encoding for task dependencies and Deep Q-learning for dynamic offloading policies in vehicular scenarios. By extracting topological features via graph neural networks and performing sequential decision-making through reinforcement learning, the algorithm significantly enhances its perception and decision-making capabilities for complex task dependencies.
2. **A critical path-driven dual-layer task priority scheduling mechanism is proposed.** Innovatively introducing critical path theory into the dynamic edge computing environment, this study designs an adaptive priority evaluation method that combines topological structure with real-time resource states, effectively addressing resource competition and sequential constraints in dependent task scheduling.
3. **A distributed offloading decision mechanism based on multi-agent collaboration is designed.** By uniformly modeling vehicles and Roadside Units (RSUs) as heterogeneous nodes in a graph, collaborative resource allocation is achieved in multi-vehicle, multi-edge-server environments, overcoming the scalability and real-time limitations of traditional single-device or centralized scheduling approaches.

This study still has several limitations: First, the current model primarily optimizes latency metrics and does not thoroughly consider multi-objective trade-offs such as energy consumption and bandwidth costs. Second, the algorithm training relies on simulation environments, and its generalization capability in real-world vehicular networking scenarios requires further validation. Additionally, the modeling of frequent handovers and task migration overhead caused by high-speed vehicle movement could be further refined.

Future work will focus on the following directions: (1) Investigating dependent task offloading mechanisms for multi-objective joint optimization to balance latency, energy consumption, and economic costs; (2) Developing lightweight online learning frameworks to enable adaptive and rapid deployment of algorithms in dynamic edge environments; (3) Exploring cross-layer task scheduling under vehicle-edge-cloud three-tier collaborative architectures to further enhance the quality of service for complex vehicular applications.

7 Acknowledgements

This research was supported by the Gansu Technological Innovation Program Project (Grant No. 24CXNA061).

References

[1] F.E. da Silva Barbosa, F.F. de Mendonça Júnior, K.L. Dias, A Platform for Cloudification of Network and Applications in the Internet of Vehicles, Transactions on Emerging Telecommunications Technologies 31(5)(2020) e3961. https://doi.org/10.1002/ett.3961

[2] D.C. Nguyen, M. Ding, P.N. Pathirana, A. Seneviratne, J. Li, D. Niyato, O. Dobre, H.V. Poor, 6G Internet of Things: A Comprehensive Survey, IEEE Internet of Things Journal 9(1)(2022) 359-383. https://doi.org/10.1109/JIOT.2021.3103320

[3] K.R. Reddy, A. Muralidhar, Machine Learning-Based Road Safety Prediction Strategies for Internet of Vehicles (IoV) Enabled Vehicles: A Systematic Literature Review, IEEE Access 11(2023) 112108-112122. https://doi.org/10.1109/ACCESS.2023.3315852
[4] Z. Wang, J. Zheng, Y. Wu, N. Mitton, A Centrality-based RSU Deployment Approach for Vehicular Ad Hoc Networks, in: Proc. 2017 IEEE International Conference on Communications, 2017. https://doi.org/10.1109/ICC.2017.7996986
[5] R. Cai, Y. Feng, D. He, Y. Xu, Y. Zhang, W. Xie, A Combined Cable-Connected RSU and UAV-Assisted RSU Deployment Strategy in V2I Communication, in: Proc. ICC 2020 - 2020 IEEE International Conference on Communications, 2020. https://doi.org/10.1109/ICC40277.2020.9149122
[6] S.-Y. Lin, C.-M. Huang, T.-Y. Wu, Multi-Access Edge Computing-Based Vehicle-Vehicle-RSU Data Offloading Over the Multi-RSU-Overlapped Environment, IEEE Open Journal of Intelligent Transportation Systems 3(2022) 7-32. https://doi.org/10.1109/OJITS.2022.3142065
[7] J. Zhao, Offloading Selection Based on Heterogeneous Utility in MEC Networks: A Coalition Formation Game-theoretic Approach, in: Proc. 2021 IEEE 6th International Conference on Computer and Communication Systems, 2021. https://doi.org/10.1109/ICCCS52626.2021.9449231
[8] D. Zhuang, Q. He, X. Chen, Z. Lin, P. Chen, Offloading Strategy for Vehicles in the Architecture of Vehicle-MEC-Cloud, in: Proc. 2022 IEEE/CIC International Conference on Communications in China, 2022. https://doi.org/10.1109/ICCCWorkshops55477.2022.9896685
[9] Md.S. Hossain, M.M.A. Hashem, An Improved Dependent Task Offloading in Device-Edge-Cloud using Favorite Predecessor Task, in: Proc. 2024 2nd International Conference on Information and Communication Technology, 2024. https://doi.org/10.1109/ICICT64387.2024.10839740
[10] E. Lee, H. Park, C. Kim, S. Lee, Offloading Dependent Tasks with Mobility in Mobile Edge Computing, in: Proc. 2022 IEEE International Conference on Communications Workshops, 2022. https://doi.org/10.1109/ICCWorkshops53468.2022.9882162
[11] Z. Niu, X.S. Shen, Q. Zhang, Y. Tang, Space-Air-Ground Integrated Vehicular Network for Connected and Automated Vehicles: Challenges and Solutions, Intelligent and Converged Networks 1(2)(2020) 142-169. https://doi.org/10.23919/ICN.2020.0009
[12] C. Anzola-Rojas, R.J. Durán Barroso, I. de Miguel, J.C. Aguado, N. Merayo, P. Fernández, R.M. Lorenzo, E.J. Abril, Distributed Task Offloading in MEC Networks for Temporary Peaks in Demand, in: Proc. 2023 IEEE Latin-American Conference on Communications, 2023. https://doi.org/10.1109/LATINCOM59467.2023.10361901
[13] J. Yin, L. Wei, Z. Liu, X. Yang, H. Sun, Y. Cheng, J. Mai, VDCM: A Data Collection Mechanism for Crowd Sensing in Vehicular Ad Hoc Networks, Big Data Mining and Analytics 6(4)(2023) 391-403. https://doi.org/10.26599/BDMA.2022.9020041
[14] F. Wu, S. Wang, J. Zhou, H. Pan, C. Zhou, W. Yang, F. Lyu, Y. Zhang, Multi-Variate Time Series Prediction of Traffic and Users for Dynamic RRH-BBU Mapping in C-RAN, IEEE Transactions on Mobile Computing 24(10)(2025) 10557-10572. https://doi.org/10.1109/TMC.2025.3570851
[15] J. Xia, Y. Liu, L. Tan, Joint Optimization of Trajectory and Task Offloading for Cellular-Connected Multi-UAV Mobile Edge Computing, Chinese Journal of Electronics 33(3)(2024) 823-832. https://doi.org/10.23919/cje.2022.00.159
[16] F. Wu, F. Lyu, H. Wu, J. Ren, Y. Zhang, X. Shen, Characterizing User Association Patterns for Optimizing Small-Cell Edge System Performance, IEEE Network 37(3)(2023) 210-217. https://doi.org/10.1109/MNET.121.2200089
[17] X. Li, J. Tang, Y. Xu, Y. Sun, Mobility-Aware Multi-Task Migration and Offloading Scheme for Internet of Vehicles, Chinese Journal of Electronics 32(6)(2023) 1192-1202. https://doi.org/10.23919/cje.2022.00.333
[18] C. Wu, Z. Huang, Y. Zou, Delay Constrained Hybrid Task Offloading of Internet of Vehicle: A Deep Reinforcement Learning Method, IEEE Access 10(2022) 102778-102788. https://doi.org/10.1109/ACCESS.2022.3206359
[19] J. Wang, P. Chen, Q. Yin, M. Yu, M. Zhang, X. Zhang, A DRL-based algorithm for Dependent Task Offloading in Multi-access Edge Computing, in: Proc. 2023 5th International Academic Exchange Conference on Science and Technology Innovation, 2023. https://doi.org/10.1109/IAECST60924.2023.10503133
[20] W. Zou, Z. Zhang, N. Wang, T. Yu, L. Tian, ReflexPilot: Startup-aware Dependent Task Scheduling based on Deep Reinforcement Learning for Edge-cloud Collaborative Computing, IEEE Transactions on Cloud Computing 13(2) (2025) 641-654. https://doi.org/10.1109/TCC.2025.3555231
[21] J. Lou, Z. Tang, W. Jia, W. Zhao, J. Li, Startup-Aware Dependent Task Scheduling With Bandwidth Constraints in Edge Computing, IEEE Transactions on Mobile Computing 23(2)(2024) 1586-1600. https://doi.org/10.1109/TMC.2023.3238868

[22] R. Zhang, L. Wu, S. Cao, X. Hu, S. Xue, D. Wu, Q. Li, Task Offloading with Task Classification and Offloading Nodes Selection for MEC-Enabled IoV, ACM Transactions on Internet Technology 22(2)(2021) 51.
https://doi.org/10.1145/3475871

[23] M.K. Aslam, M.A. Javed, G. Srivastava, A. Ranjha, A.N. Alvi, B.Q. Ali, J. Mirza, Efficient Task Offloading in NOMA-Assisted Vehicular Fog Computing for Customer Applications, IEEE Transactions on Consumer Electronics 71(2)(2025) 6521-6527.
https://doi.org/10.1109/TCE.2024.3509274

[24] S. Zhang, N. Yi, Y. Ma, A Survey of Computation Offloading With Task Types, IEEE Transactions on Intelligent Transportation Systems 25(8)(2024) 8313-8333.
https://doi.org/10.1109/TITS.2024.3410896

[25] A.A. Al-Habob, A. Ibrahim, O.A. Dobre, A.G. Armada, Collision-Free Sequential Task Offloading for Mobile Edge Computing, IEEE Communications Letters 24(1)(2020) 71-75.
https://doi.org/10.1109/LCOMM.2019.2948179

[26] J. Zhang, J. Chen, X. Bao, C. Liu, P. Yuan, X. Zhang, S. Wang, Dependent Task Offloading Mechanism for Cloud–Edge-Device Collaboration, Journal of Network and Computer Applications 216(2023) 103656.
https://doi.org/10.1016/j.jnca.2023.103656

[27] X. He, R. Jin, H. Dai, Multi-Hop Task Offloading With On-the-Fly Computation for Multi-UAV Remote Edge Computing, IEEE Transactions on Communications 70(2)(2022) 1332-1344.
https://doi.org/10.1109/TCOMM.2021.3129902

[28] Y. Cui, L. Du, P. He, D. Wu, W.R. Wang, Cooperative Vehicles-Assisted Task Offloading in Vehicular Networks, Transactions on Emerging Telecommunications Technologies 33(7)(2022) 4472.
https://doi.org/10.1002/ett.4472

[29] C.-T. Mo, J.-H. Chen, W. Liao, Graph Convolutional Network Augmented Deep Reinforcement Learning for Dependent Task Offloading in Mobile Edge Computing, in: Proc. 2023 IEEE Wireless Communications and Networking Conference, 2023.
https://doi.org/10.1109/WCNC55385.2023.10119034

[30] J. Chen, Z. Wu, Dynamic Computation Offloading With Energy Harvesting Devices: A Graph-Based Deep Reinforcement Learning Approach, IEEE Communications Letters 25(9)(2021) 2968-2972.
https://doi.org/10.1109/LCOMM.2021.3094842

[31] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, X. Wang, Multitask Offloading Strategy Optimization Based on Directed Acyclic Graphs for Edge Computing, IEEE Internet of Things Journal 9(12)(2022) 9367-9378.
https://doi.org/10.1109/JIOT.2021.3110412

[32] H. Ye, J. Li, Q. Lu, Deep Reinforcement Learning for Dependent Task Offloading in Multi-Access Edge Computing, IEEE Access 12(2024) 166281-166297.
https://doi.org/10.1109/ACCESS.2024.3494775

[33] S. Yang, L. Zhang, L. Cui, Q. Dong, W. Xiao, C. Luo, RLCS: Towards a Robust and Efficient Mobile Edge Computing Resource Scheduling and Task Offloading System Based on Graph Neural Network, Computer Communications 206(2023) 38-50.
https://doi.org/10.1016/j.comcom.2023.04.020