Actor-Critic Convolutional Neural Network-Based Centralized Computation Offloading for Vehicular Edge Networks

Xuan Ouyang [1] XueXue Zhang [2]

[1] Institute of Forest Resource Information Techniques, Chinese Academy of Forestry, Haidian District 100091, Beijing, China

[21] Beijing Zhongtian Xingye Cultural Development Co., Ltd., Haidian District 100083, Beijing, China;

ABSTRACT Efficient computation offloading is essential for addressing the growing demand for low-latency, computation-intensive applications in vehicular edge networks. Existing approaches, including traditional optimization techniques and reinforcement learning-based methods, often struggle to cope with the highly dynamic network environments and the complex task dependencies characterized by Directed Acyclic Graphs (DAGs). To address these challenges, this paper proposes a novel centralized Actor-Critic Convolutional Neural Network-based Offloading Computation (ACCOC) algorithm. By leveraging the Actor-Critic framework enhanced with convolutional layers, ACCOC efficiently handles high-dimensional task offloading decisions, balancing delay and energy consumption under dynamic network conditions. The proposed algorithm is evaluated in a realistic vehicular edge computing environment with varying numbers of users, bandwidth availability, and edge server computational capabilities. Simulation results reveal that ACCOC achieves superior performance compared to baseline methods, including a non-convolutional Actor-Critic algorithm, Deep Q-Network (DQN), and random or fully local execution strategies. Specifically, ACCOC demonstrates faster convergence and higher system utility, achieving an average reward of 0.78 compared to 0.72 for the non-convolutional baseline. Furthermore, the algorithm maintains robust performance across diverse parameter settings, highlighting its scalability and adaptability.

Keywords Vehicular Edge Computing, Computation Offloading, Actor-Critic, Convolutional Neural Networks, Reinforcement Learning, System Optimization

I. INTRODUCTION

The continuous advancement of intelligent vehicular applications, including autonomous driving, collaborative traffic management, and immersive infotainment systems, has dramatically increased the computational and communication demands on vehicular networks [1]. Vehicular Edge Computing (VEC), by deploying edge servers near vehicles, emerges as a pivotal solution to alleviate the limitations of centralized cloud computing, enabling low-latency task execution and reduced network congestion [2]. Despite its potential, realizing efficient computation offloading in VEC remains a formidable challenge due to the dynamic nature of vehicular environments and the heterogeneous demands of computation-intensive tasks.

(1)Challenges in Computation Offloading

Computation offloading decisions in VEC networks involve complex trade-offs between delay, energy consumption, and resource allocation efficiency [3]. The problem becomes particularly challenging when task dependencies, represented as Directed Acyclic Graphs (DAGs), are considered [4]. Traditional approaches, such as heuristic algorithms and convex optimization, often fail to cope with the dynamic and high-dimensional nature of VEC environments [5]. Recent efforts leveraging reinforcement learning (RL) show promise in addressing these challenges [6]. However, conventional RL algorithms, including Deep Q-Networks (DQN) and Actor-Critic methods, face scalability limitations and slow convergence when applied to DAG-based task offloading in large-scale vehicular networks.

(2) Motivation and Contributions

To address these challenges, we propose a centralized Actor-Critic Convolutional Neural Network-based Offloading Computation (ACCOC) algorithm, tailored for VEC networks. The motivation stems from two key observations: 1. High-dimensional task dependencies: DAG-based tasks require offloading strategies capable of capturing intricate interdependencies among subtasks, which traditional RL approaches fail to effectively address.

2. Dynamic vehicular environments: The rapidly changing conditions in VEC networks necessitate algorithms that can adapt to varying user densities, bandwidth availability, and edge server capabilities. Building on these insights, the contributions of this paper are summarized as follows:

Novel Algorithm Design: We develop the ACCOC algorithm that integrates convolutional neural networks (CNNs) into the Actor-Critic framework, enabling efficient representation and processing of DAG-based task dependencies.

Comprehensive Performance Evaluation: The algorithm is rigorously evaluated in simulated VEC scenarios under diverse conditions, including varying numbers of users, bandwidth, and edge server computational capabilities.

Superior Performance: Simulation results demonstrate that ACCOC outperforms state-of-the-art methods, including non-convolutional Actor-Critic algorithms, DQN, and heuristic baselines, in terms of convergence speed, average reward, and adaptability.

The remainder of this paper is organized as follows: Section 2 reviews related work on computation offloading in VEC networks, focusing on traditional optimization methods and RL approaches. Section 3 presents the system model, including the DAG-based task representation and the network resource constraints. Section 4 details the proposed ACCOC algorithm, highlighting its design and implementation. Section 5 provides simulation results and performance analysis, demonstrating the advantages of the proposed algorithm. Section 6 discusses potential extensions and future research directions. Section 7 concludes the paper by summarizing the key findings and contributions.

II. Related Work

(1)Heuristic and Convex Optimization Approaches

Heuristic and convex optimization methods have historically played a significant role in tackling task offloading in VEC networks. Early works formulated offloading problems as optimization models aiming to minimize task delay or energy consumption. For example, convex programming techniques were employed to jointly optimize offloading decisions and resource allocation under static network conditions [7]. While effective for small-scale networks or scenarios with predictable dynamics, these methods are inherently limited by their reliance on simplified assumptions about network behavior. The high computational overhead and lack of adaptability make them unsuitable for real-time decision-making in highly dynamic vehicular environments.

(2) Reinforcement Learning in Task Offloading

RL has gained traction as a more adaptable alternative to traditional optimization methods. RL-based algorithms can learn task offloading policies through interaction with the environment, allowing them to adapt to dynamic network states. Among these, DQN have been widely studied for resource management and task scheduling. For instance, some studies applied DQN to optimize offloading decisions under varying vehicular mobility patterns and wireless conditions, achieving better adaptability compared to static optimization techniques [8]. However, DQN's reliance on discrete action spaces and limited capacity for capturing interdependencies among subtasks often leads to suboptimal performance in scenarios involving complex DAG task structures.

Actor-Critic (AC) methods have shown promise in addressing some of DQN's limitations. By decoupling the policy and value functions, AC methods enable more efficient exploration of continuous or hybrid action spaces, making them well-suited for dynamic task offloading. Recent advancements have further integrated neural networks into AC frameworks to enhance their generalization capability and convergence speed [9]. Nevertheless, existing AC-based solutions often lack the specific mechanisms required to handle DAG tasks' hierarchical and sequential dependencies, which are common in VEC systems.

(3)Centralized Offloading Strategies

Centralized task offloading strategies provide a unified framework for making decisions across multiple tasks and nodes, offering the potential for globally optimal solutions [10]. These strategies leverage centralized controllers to gather system-wide information and execute task scheduling decisions based on comprehensive network states. While centralized approaches can theoretically achieve superior performance, they are often challenged by communication overhead and computational scalability in large-scale vehicular environments. Moreover, the task dependencies inherent in DAG models remain underexplored, limiting the effectiveness of centralized algorithms in real-world scenarios.

This paper bridges the aforementioned gaps by introducing the Actor-Critic Convolutional Offloading Algorithm (ACCOC), which leverages CNNs to effectively encode DAG task structures within a single-agent RL framework. Compared to traditional RL approaches, ACCOC achieves faster convergence and superior system utility, as validated through extensive simulations.

III. System Model

We describe the VEC system model, which comprises the network topology, task structure, and optimization objectives. Our model is specifically designed to capture the hierarchical dependencies and computational constraints in dynamic vehicular environments

(1)Network Topology

Shown in Figure 1, the VEC system consists of I vehicular terminals (VTs) and M edge servers (ESs), as illustrated in Figure 1. Each VT is equipped with limited computing and communication resources, while ESs are deployed at roadside units (RSUs) to provide edge computing capabilities.

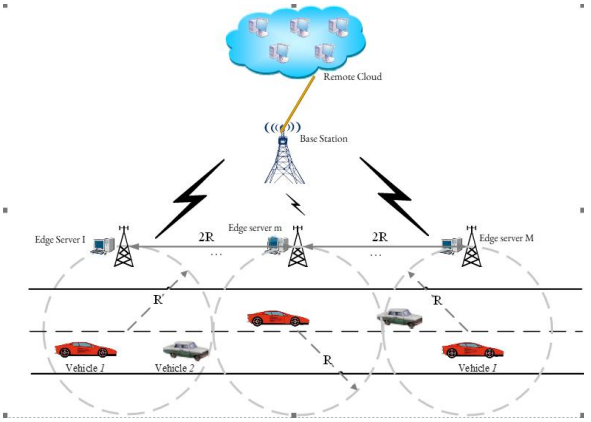

Figure 1: Scenario Model Illustration

Vehicular Terminals: Each VT generates computational tasks at discrete time slots. Tasks can either be processed locally or offloaded to ESs.

Edge Servers: Each ES has sufficient computing resources to handle multiple offloaded tasks simultaneously but is subject to latency and bandwidth constraints due to shared network resources. The communication channel between VTs and ESs is modeled as a single-hop wireless link. The link quality is influenced by factors such as vehicular mobility and signal interference.

(2)Task Model

The computational task generated by each VT is represented as a directed acyclic graph (DAG), $G = (V, E)$, where:

- $V = \{v_1, v_2, \ldots, v_{|V|}\}$ denotes the set of subtasks, with each v_j requiring a specific amount of data transmission d_j and computational resources c_j.
- $E = \{(v_i, v_j)\}$ represents the dependencies among subtasks, ensuring that a subtask v_j cannot commence until its predecessor v_i is completed.

Each DAG task must be completed within a strict deadline $T_{\max}$, accounting for both local processing and offloading-induced transmission delays. The execution options for a subtask include:

- Local Execution: Processed on the VT's onboard CPU with power consumption proportional to the computation load.
- Edge Offloading: Offloaded to an ES, which involves an uplink transmission delay and computation delay at the ES.

(3) Resource Model

- Computation Resources: Local computation capacity of each VT is f_l (in GHz); Computation capacity of each ES is f_m (in GHz), shared among all offloaded tasks.
- Communication Resources: Each VT has a transmission power p_i; The available bandwidth B is shared among all VTs, and the uplink transmission rate is governed by the Shannon-Hartley theorem:

$$R_i = B \log_2 \left(1 + \frac{p_i h_i}{\sigma^2}\right),$$

where h_i is the channel gain and σ^2 is the noise power.

(4)Energy and Latency Analysis

The total cost of completing a DAG task includes both energy consumption and latency, defined as follows:

- Energy Consumption: The energy consumption for subtask execution includes local computation energy and transmission energy for offloaded tasks:

$$E = \sum_{j \in V} E_{\text{local},j} + \sum_{j \in V_{\text{offloaded}}} E_{\text{transmit},j},$$

where $E_{\text{local},j} = K c_j f_l^2$ and $E_{\text{transmit},j} = \frac{d_j}{R_i} p_i$.

- Latency: The total latency includes local computation delay, transmission delay, and ES computation delay:

$$T = \sum_{j \in V} T_{\text{local},j} + \sum_{j \in V_{\text{offloaded}}} \left(T_{\text{transmit},j} + T_{\text{edge},j}\right)$$

where $T_{\text{local},j} = \frac{c_j}{f_l}$, $T_{\text{transmit},j} = \frac{d_j}{R_i}$, **and** $T_{\text{edge},j} = \frac{c_j}{f_m}$

3.5 Optimization Problem

The objective is to minimize the weighted sum of energy consumption and latency while ensuring task completion within the deadline $T_{\max}$. This can be formulated as:

$$\min \alpha T + \beta E$$

subject to the following constraints: (1) Dependency constraints: Predecessor subtasks must complete before their successors begin; (2) Resource constraints: The total computation and communication resources allocated must not exceed the available capacities. (3) Deadline constraint: $T \le T_{\max}$.

The problem is a NP-hard problem, which is computationally intractable for real-time decision-

making. Therefore, we propose a centralized RL-based solution using the ACCOC to address these challenges.

VI. Centralized Computation Offloading Algorithm

In the multi-user, multi-server offloading scenario, terminal devices with limited computational resources cannot provide low-latency services. By offloading tasks to nearby edge servers, the computational burden on terminals is reduced, significantly decreasing service delay. Based on the heuristic of obstacle recognition tasks, many applications consist of multiple dependent subtasks. Leveraging the parallelism of subtasks can reduce the execution latency of the application. The execution sequence of these subtasks is determined by the task scheduling priority analysis method described in Section 3.

By analyzing the data information and dependency relationships between subtasks, and adjusting the order of parallel subtasks based on a topological sorting described by a Directed Acyclic Graph (DAG), the execution latency is minimized. In this section, we propose a centralized computation offloading algorithm for tasks that have been sorted based on their priority.

Considering the vehicle-edge computing model defined in Section 3, a Software Defined Network (SDN) framework is used to obtain offloading decisions for each terminal task under the premise of full user information sharing. The SDN collects user and server information, including location, communication bandwidth, computing capacity, and personal preferences, to centrally control the execution strategy of terminal tasks. The SDN is modeled as a RL agent, where the collected information forms the environment. The agent interacts with the environment, and the action taken (offloading decision) is obtained based on the state information read from the environment. A reward function is defined to evaluate state-action pairs based on the optimization objective.

The Actor-Critic algorithm integrates both policy gradient and value functions, enabling it to capture potential rewards from the environment's state. The value function guides the policy parameter updates, and the algorithm supports single-step updates. This section formulates the task offloading process as a MDP, and the offloading decision problem is equivalently replaced by a CNN-based Actor-Critic algorithm. Additionally, an experience replay technique is introduced to improve model training efficiency.

(1)MDP Core Element Analysis

The vehicle task offloading decision process is described as a MDP. In this part, the definitions of the three core RL elements—state, action, and reward—are provided as follows:

State S_t^{curr}: The state S_t^{curr} is defined in two parts: (1) User task information and the offloading decisions for the previous subtasks $S_t^{curr\_1}$: Task information includes the subtask priority sequence $V_i^{'}$ and the set of predecessor and successor tasks for the current subtask ps_i^{curr}. (2) Basic information of the server and the user $S_t^{curr\_2}$: This includes the terminal's location $dpos$ and the server's location $spos$.

Thus, the state S_t^{curr} can be represented as:

$$S_t^{curr} = [S_t^{curr\_1}, S_t^{curr\_2}],$$

where $S_t^{curr\_1}$ and $S_t^{curr\_2}$ are defined as:

$$S_t^{curr\_1} = \begin{bmatrix} V_0^{'} & ps_0^{curr} & | & A_0^k \\ \cdots & \cdots & | & \cdots \\ V_i^{'} & ps_i^{curr} & | & A_i^k \\ \cdots & \cdots & | & \cdots \\ V_{I-1}^{'} & ps_{I-1}^{curr} & | & A_{I-1}^k \end{bmatrix}$$

$$S_t^{curr\_2} = [dpos, spos, \ldots]$$

Action A_t^{curr}: At time slot *t*, the offloading decisions for all terminal applications' *curr*-th subtasks form the action set A_t^{curr}:

$$A_t^{curr} = \left(a_{1,curr}^t, \ldots, a_{i,curr}^t, \ldots, a_{I,curr}^t\right) \quad i\epsilon[1, I],$$

where $a_{i,curr}^t$ represents the offloading decision for vehicle i's *curr*-th subtask at time slot *t*.

Reward R_t^{curr}: The reward function plays a critical role in guiding offloading decisions. For each action, the reward will steer the system towards optimizing the task offloading strategy. To improve system performance, the reward is set as a function of both system latency and energy consumption.

From the above analysis, the execution time for each subtask $t_{i,j}$ is expressed as:

$$t_{i,j} = \begin{cases} t_{i,j}^l & \text{if } a_{i,j}^t = 0 \\ t_{i,j}^m & if\ a_{i,j}^t = \mathrm{m}, \mathrm{m}\epsilon M \\ t_{i,j}^{cc} & \text{if } a_{i,j}^t = -1 \end{cases}.$$

Similarly, the energy consumption $E_{i,j}$ for each subtask is represented as:

$$E_{i,j} = \begin{cases} \mathrm{E}_{i,j}^l & \text{if } a_{i,j}^t = 0 \\ E_{i,j}^m & if\ a_{i,j}^t = \mathrm{m}, \mathrm{m}\epsilon M \\ E_{i,j}^{cc} & \text{if } a_{i,j}^t = -1 \end{cases}.$$

The increment in time $\triangle T_{i,j}$ and energy $\triangle E_{i,j}$ during the DAG task offloading process are as follows:

$$\triangle T_{i,j} = EST\left(v_{i,j+1}\right) - EST\left(v_{i,j}\right),$$
$$\triangle E_{i,j} = ESE\left(v_{i,j+1}\right) - ESE\left(v_{i,j}\right).$$

The start time $ST_{i,j}$ and finish time $FT_{i,j}$ for each subtask are computed using:

$$ST_{i,j} = \begin{cases} 0 & if\ v_{i,j}\ is\ entry\ task \\ FT_{i,j^{'}} & otherwise \end{cases},$$
$$FT_{i,j} = ST_{i,j} + \triangle T_{i,j}.$$

Likewise, the energy consumption at the start $SE_{i,j}$ and finish $FE_{i,j}$ are given by:

$$SE_{i,j} = \begin{cases} 0 & if\ v_{i,j}\ is\ entry\ task \\ FE_{i,j^{'}} & otherwise \end{cases},$$
$$FE_{i,j} = SE_{i,j} + \triangle E_{i,j}.$$

Thus, the reward R_t^{curr} is defined as:

$$R_t^{curr} = \alpha \frac{\sum_{i=1}^{I}\sum_{j=1}^{curr} t_{i,j}^{l} - \sum_{i=1}^{I}\sum_{j=1}^{curr} FT_{i,j}}{\sum_{i=1}^{I}\sum_{j=1}^{curr} t_{i,j}^{l}} + \beta \frac{\sum_{i=1}^{I}\sum_{j=1}^{curr} E_{i,j}^{l} - \sum_{i=1}^{I}\sum_{j=1}^{curr} FE_{i,j}}{\sum_{i=1}^{I}\sum_{j=1}^{curr} E_{i,j}^{l}},$$

where the latency weight factor α and energy weight factor β satisfy:

$$\alpha + \beta = 1.$$

(2)Network Model Construction

In this part, a CNN-based Actor-Critic model is used to approximate both the Actor and Critic networks. The Actor's network structure consists of an input layer, I agent frameworks, each containing 2 convolutional layers, pooling layers, a flatten layer, 2 fully connected layers, and a softmax layer. The input layer has the same dimension as the state, and the softmax layer outputs the action probability distribution for the current state. The ReLU activation function is used in convolutional layers to enhance the model's nonlinear fitting capability. The Critic network structure consists of an input layer, 2 convolutional layers, pooling layers, a flatten layer, and 2 fully connected layers. The Critic network outputs the evaluation value for the current state.

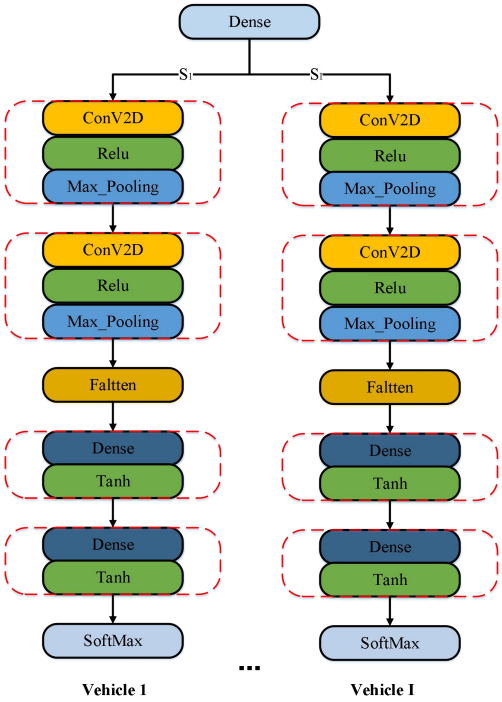

Figure 2: Actor network structure

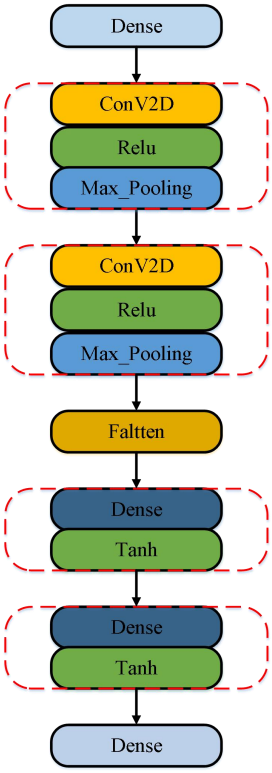

Figure 3: Critic network structure

Because the state space is large, directly using fully connected neural networks for RL training is inefficient. To address this, a convolutional network is introduced into the Actor-Critic model to extract DAG task information from $S_t^{curr\_1}$ and feed it into the fully connected layers along with location and server information from $S_t^{curr\_2}$.

(3)Training Process

The training process is based on experience replay, which stores experience samples $\{S_t^{curr}, A_t^{curr}, R_t^{curr}, S_t^{curr+1}\}$. Once the experience buffer is full, random samples are drawn to update the Actor and Critic networks, helping the model learn to avoid overfitting to specific experiences. Shown in Algorithm 1, the training steps are as follows:

1.Initialization: Randomly initialize parameters for both Actor and Critic networks.

2.Sampling: Gather current state information and interactions, storing experiences sequentially.

3.Training: After the buffer fills, sample mini-batches for network updates.

4.Termination: Continue until convergence or the maximum number of iterations is reached.

Algorithm 1: Actor Critic Centralized Offloading Algorithm basec
Neural Network (ACCOC)

Input: I, M, $G_i = (V_i, E_i)$, R, B, σ^2, K, f_i, f_m, p_i, α, β, $Batcl$
Output: Subtasks offloading decisions for different vehicles
Initialize: network parameters θ, ω, and replay buffer D
for each episode **do**
  Schedule subtask by TSPA
  Obtain the state S_t^{curr} from the current environment
  for t=1, 2, …, T **do**
    Select action A_t^{curr}
    Get reward R_t^{curr} by (4.16) and next state S_t^{curr+1} after
    Store $\left(S_t^{curr}, A_t^{curr}, R_t^{curr}, S_t^{curr+1}\right)$ into D
    The next state S_t^{curr+1} replaces the current state S_t^{curr}
    if pointer % D == 0 **then**
      Randomly select a $Batch$ of samples from D
      Update critic network parameter ω and actor network
      Rewrite the replay buffer D
    end if
  end for
end for

.

V. Simulation Results and Analysis

(1)Simulation Environment and Parameter Settings

In this part, the system environment and centralized computation offloading algorithm are implemented using the Pytorch framework in a Python environment. The simulation environment consists of 5 vehicle terminals and 3 edge servers. In each time slot, all terminals generate fine-grained tasks, which are offloaded based on the task scheduling priority analysis and the centralized computation offloading algorithm. The main simulation parameters and training hyperparameters are listed in Table I.

Table I: Simulation Environment Parameters and Training Hyperparameters

Parameter	Value
Number of Vehicles I	5
Number of Edge Servers M	3
Vehicle Speed	17m/s
Vehicle Computing Capacity f_l	0.5GHz
Edge Server Computing Capacity f_m	5GHz
Vehicle Transmission Power p_i	0.1Watt
Bandwidth B	100MHz
Signal Coverage Range R	200m
Number of Subtasks $\|V_i\|$	[8,10]
Subtask Data Size $d_{i,j}$	[100,500]KB
Subtask CPU Cycles $c_{i,j}$	[10,50]MC
Noise Power σ^2	-100dbm
Latency Weight α	0.5
Energy Weight β	0.5
Buffer Capacity D	100
Training Batch Size	32
Actor Learning Rate	0.005
Critic Learning Rate	0.01
Discount Factor	0.9

To simplify the experiment, it is assumed that all terminals and edge servers have the same computing capacity. The vehicle computing power f_l, edge server computing power f_m, transmission power p_i, and effective capacitance switch K values are referenced from literature [11]. The subtask data size and required CPU cycles are randomly generated using a uniform distribution as specified in Table I. The experience replay buffer is set with a capacity of 100, and when the buffer reaches this capacity, 32 samples are randomly selected for training and updating the network parameters.

In the above simulation parameter setup, the subtask information within the DAG is set within a large range to ensure the ACCOC algorithm has general applicability. To further validate the effectiveness of the algorithm, the performance under varying user numbers, bandwidth, and MEC computing capacity is analyzed in Section 5.2.

(2)Performance Analysis

2.1 Convergence Evaluation

We evaluate and analyze the performance of the proposed ACCOC algorithm based on the simulation results. The ACCOC algorithm is compared with the ACCOFC algorithm, DQN algorithm, random (RANDOM) algorithm, and all-local (AL) algorithm. The RANDOM algorithm represents a scenario where terminal tasks are randomly selected to be processed locally or offloaded. The AL algorithm represents the scenario where all terminal tasks are processed locally.

Figure 4 shows the average reward value for each round of training, where the x-axis represents the number of training rounds, and the y-axis represents the average reward value per round. In this paper, a training round refers to the process of handling a single DAG task. Based on the definition of the reward value, a reward value less than 0 indicates that the task offloading performance is worse than local execution, while a reward value equal to 0 means that the algorithm has no effect. The goal is to have a reward value greater than 0, with higher values being better. This section explores the convergence performance of different algorithms.

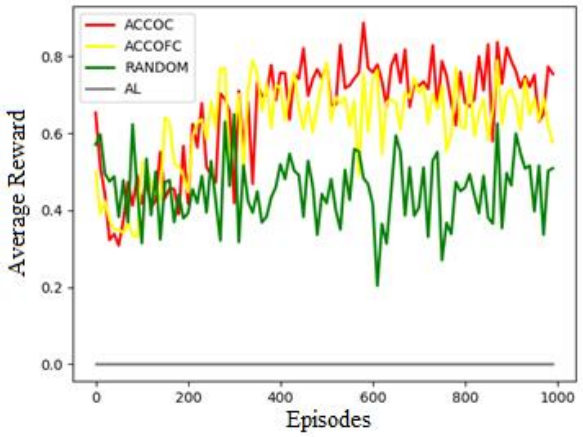

Figure 4: System Performance under Different Training Rounds

From Figure 4, the proposed ACCOC algorithm's reward value converges to approximately 0.78, the ACCOFC algorithm converges to 0.72, and the RANDOM algorithm approaches 0.44. Based on the reward value definition, the AL algorithm consistently produces a reward value of 0. The proposed ACCOC algorithm outperforms the ACCOFC, RANDOM, and AL algorithms in terms of convergence speed and the final value, achieving the best system utility.

2.2 Impact of Number of Users on System Performance

Next, we experiment with the relationship between the number of users and system performance. The x-axis in Figure 5 represents the number of terminal users, and the y-axis represents the average system loss after convergence for the three algorithms. This experiment evaluates how the average system loss changes as the number of users increases. The average system loss refers to the weighted sum of the actual delay and energy consumption for a terminal user to complete a DAG task in the mobile edge computing system.

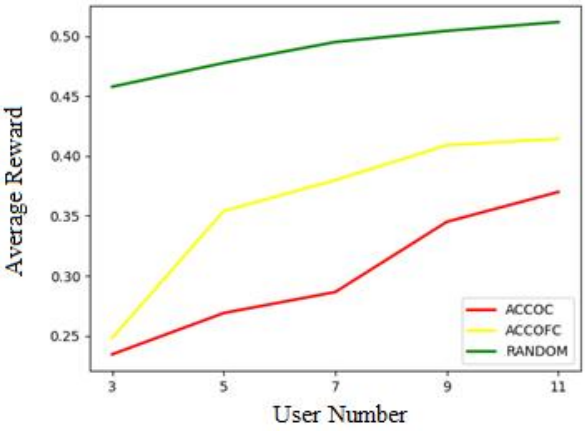

Figure 5: Average System Loss under Different Numbers of Users

From Figure 5, it can be observed that the system loss for all three algorithms increases as the number of users grows. This is because an increased number of users leads to more terminals competing for limited channel and computing resources. Among the three algorithms, the proposed ACCOC algorithm results in the lowest consumption, while the ACCOFC algorithm shows a significantly higher average system energy consumption.

2.3 Impact of Bandwidth on System Performance

Next, we analyze the relationship between communication bandwidth and system performance. The x-axis in Figure 6 represents the communication bandwidth available at the edge servers, and the y-axis represents the average system loss after convergence for the algorithms. To more effectively compare the impact of bandwidth on system loss, the number of subtasks generated for each time slot is fixed at 10, with a subtask data size of 300 KB and required CPU cycles set to 30 MC.

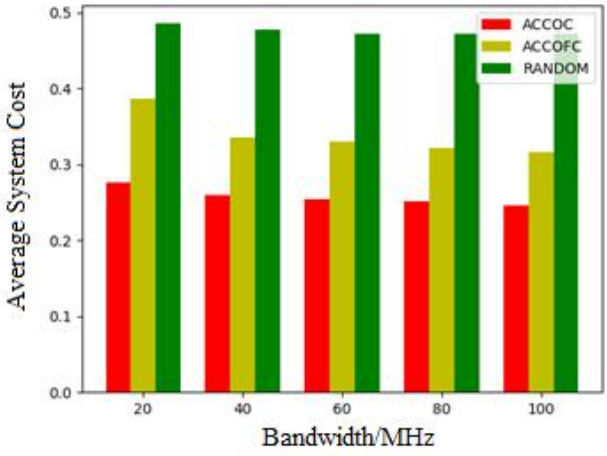

Figure 6: Average System Loss under Different Bandwidths

From Figure 6, it can be seen that as the communication bandwidth of the edge servers increases, the average system loss for the ACCOC, ACCOFC, and RANDOM algorithms decreases. This is because the increased uplink bandwidth for offloading terminal tasks to edge servers reduces the task upload time and energy consumption. The proposed ACCOC algorithm shows the lowest and most stable system loss.

2.4 Impact of Edge Server Computing Power on System Performance

Finally, we investigate the impact of edge server computing power on system performance. The x-axis in Figure 7 represents the computing power of the edge servers, with values set at 1 GHz, 3 GHz, 5 GHz, 7 GHz, and 9 GHz. The y-axis represents the average system loss of the algorithms at different computing powers. To minimize random factors, the subtask data size is fixed at 300 KB, and the required CPU cycles are set to 30 MC.

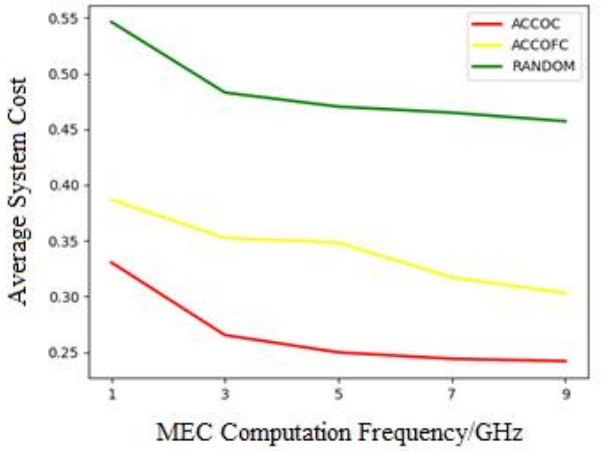

Figure 7: Average System Loss under Different Edge Server Computing Powers

From Figure 7, it is observed that as the edge server computing power increases, the average system loss for all three algorithms decreases. This is because the time required for terminal tasks to be processed on the edge server decreases as its computing capacity improves. Moreover, the proposed ACCOC algorithm consistently achieves the lowest average system loss across different parameters and remains more stable than the ACCOFC algorithm.

VI. Conclusion

In this paper, we proposed a novel ACCOC (Adaptive Centralized Computation Offloading Control) algorithm for optimizing task offloading in mobile edge computing systems. The proposed algorithm efficiently balances the computation and communication resources in a vehicular network with edge servers, considering the dynamic task requirements and resource constraints.

Through extensive simulation experiments, we evaluated the performance of the ACCOC algorithm under various conditions, including different numbers of users, bandwidth, and edge server computing power. The results demonstrated that the ACCOC algorithm outperforms several baseline algorithms, such as the ACCOFC, DQN, RANDOM, and AL algorithms, in terms of convergence speed, system utility, and overall system loss. In future work, we aim to explore decentralized approaches and further improve the scalability and robustness of the algorithm in large-scale systems.

REFERENCES

[1] Liu H, Huang W, Kim D I, et al. Towards Efficient Task Offloading with Dependency Guarantees in Vehicular Edge Networks through Distributed Deep Reinforcement Learning[J]. IEEE Transactions on Vehicular Technology, 2024.

[2] Wang J, Zhao H, Liu H, et al. A distributed vehicle-assisted computation offloading scheme based on drl in vehicular networks[C]//2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE, 2022: 200-209.

[3] Wang Y, Zhao H, Liu H, et al. A dependency-aware offloading algorithm based on deep reinforcement learning for vehicular networks[C]//2021 International Conference on Space-Air-Ground Computing (SAGC). IEEE, 2021: 63-69.

[4] Geng L, Zhao H, Liu H, et al. Deep reinforcement learning-based computation offloading in vehicular networks[C]//2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom). IEEE, 2021: 200-206.

[5] Liu H, Zhao H, Geng L, et al. A policy gradient based offloading scheme with dependency guarantees for vehicular networks[C]//2020 IEEE Globecom Workshops (GC Wkshps. IEEE, 2020: 1-6.

[6] Sun H, Wang J, Yong D, et al. Deep Reinforcement Learning-Based Computation Offloading for Mobile Edge Computing in 6G[J]. IEEE Transactions on Consumer Electronics, 2024.

[7] Liu B, Liu C, Peng M. Computation offloading and resource allocation in unmanned aerial vehicle networks[J]. IEEE Transactions on Vehicular Technology, 2022, 72(4): 4981-4995.

[8] Chi J, Zhou X, Xiao F, et al. Task Offloading via Prioritized Experience-based Double Dueling DQN in Edge-assisted IIoT[J]. IEEE Transactions on Mobile Computing, 2024.

[9] Qiu B, Wang Y, Xiao H, et al. Deep Reinforcement Learning-Based Adaptive Computation Offloading and Power Allocation in Vehicular Edge Computing Networks[J]. IEEE Transactions on Intelligent Transportation Systems, 2024.

[10] Yan J, Zhao X, Li Z. Deep Reinforcement Learning Based Computation Offloading in UAV-Assisted Vehicular Edge Computing Networks[J]. IEEE Internet of Things Journal, 2024.

[11] Tong Z, Deng X, Ye F, et al. Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment[J]. Information Sciences, 2020, 537: 116-131.