2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD) | 979-8-3503-4918-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/CSCWD61410.2024.10580013

Lyapunov-guided Deep Reinforcement Learning for Vehicle task Stable offloading

Ziyang Huang
Anhui University
Hefei, China
e22301367@stu.ahu.edu.cn

Yanming Chen
Anhui University
Hefei, China
cym@ahu.edu.cn

Yiwen Zhang
Anhui University
Hefei, China
zhangyiwen@ahu.edu.cn

***Abstract*—Cellular vehicular-to-everything (C-V2X), a critical Internet of Vehicles (IOV) technology is promised to be enhanced and strengthened to improve road traffic safety and achieve intelligent transportation in the 5G era. However, computation-intensive and latency-sensitive computation tasks of autonomous driving have created a great challenge for computation and storage-limited vehicles. Vehicular edge computing (VEC) is envisioned as a promising approach to processing the explosive computation tasks of vehicular users (VU). In the VEC system, each VU allocates to process partial tasks through offloading and the remaining tasks through local execution. In practical scenarios, the number of vehicles and the arrival of vehicle tasks are random, leading to a highly complex environment for VEC systems. To solve this problem, we propose a novel framework, named LYDDPG, that combines the advantages of Lyapunov optimization and deep reinforcement learning (DRL) to ensure the stability of the system during task offloading.**

***Index Terms*—Vehicular edge computing, Deep reinforcement learning, Lyapunov function, Task offloading**

I. Introduction

With the development of 5G mobile communication network technology and the continuous enhancement of social communication infrastructure, an era of interconnected everything is approaching. At the same time, the significant improvement in network connection speed and object perception ability [1], and the continuous updates of sensor technology have led to the flourishing development of cellular vehicular-to-everything (C-V2X) [2]. For example, modern cars such as the Tesla Model X is already equipped with eight cameras, twelve ultrasonic radars, and a millimeter wave radar, the data calculation requirements are also getting higher and higher.

At the same time, in vehicle applications such as multi media entertainment, real-time navigation, assisted driving, and driving behavior monitoring have been widely applied to smart vehicle users (SVU) [3]. These applications require a large amount of data and computing resources, thus placing a heavy computational burden on vehicle user (VUs) with limited computing power [4] [5]. Traditional cloud computing requires vehicles to transmit computing data to the cloud service with strong computing power at the remote end, which increases more load pressure, delay, and energy consumption [6]. Vehicular edge computing (VEC) is a promising way to relieve system burden [7] [8] [9]. This paper studies a VEC system that allows multiple vehicles with different task sizes perform task offloading simultaneously, proposed a centralized offloading decision algorithm based on an improved deep deterministic policy gradient (NLDDPG) algorithm [16] improvement for VEC, namely Lyapunov-guided Deep Reinforcement Learning (LYDDPG). We consider the influence of vehicle motion, speed, position, signal interference, and tasks of different sizes. Use reinforcement learning to complete the mapping from the vehicle state space to the task offloading action. As the number of vehicles increases, so does the competition for VEC server resources, which can lead to instability in the system. We use the Lyapunov function [17] to ensure the stability of the task and energy queues, ensuring the system's convergence. The main contributions can be summarized as follows:

- For the VEC environment, the reinforcement learning method is adopted to make different task offloading decisions according to various state spaces, and network parameters are constantly adjusted through learning to ensure that the system's joint optimization of delay and energy reaches the optimal.
- Multiple vehicles needing task offloading share the resources of the edge server, and the Lyapunov function is used to keep the stability of the task queue and the energy queue, preventing the edge server from becoming unstable due to excessive load. The experiment shows that with the increasing number of vehicles needing task offloading, the system load can also remain stable.
- Considering the changes in the traffic environment, such as the difference in the number of vehicles, the change in vehicle position, the difference in task size, the difference in vehicle speed, and the impact of signal interference on data transmission rate during transmission.

In recent years, vehicle task offloading has been widely studied, and a series of solutions have been proposed to improve the execution delay or energy consumption, or to jointly optimize the two as much as possible.

Luo et al. [10] proposed a dynamic programming algorithm to minimize the latency during task offloading. Huang et al. [11] proposed a distributed reputation management system to improve the network efficiency in vehicle edge computing and reduce data transmission delay. Guo jointly optimized the allocation of computing resources for radio resources in [12] to

minimize energy consumption under delay constraints during task offloading. Han et al. [13]used heuristic algorithms to make task offloading decisions to minimize task latency and energy consumption. Rahabri et al. [14] proposed a greedy approach that uses a backpack-based scheduling algorithm (GKS) to solve the task assignment problem, the greedy algorithm continuously approximates its solution to the optimal solution through continuous iteration. The above works are mainly applicable to the static state space, and the environment is constantly changing in the vehicle environment, so dynamic task offloading algorithms have emerged as the times required. Gao proposed a distributed reinforcement learning method in [15], which allows each vehicle to make unloading decisions independently, ignoring the load of the whole system. Hu and Huang proposed a task offloading scheme based on deep reinforcement learning in [16], the VEC system makes different decisions based on different states and moving vehicles can offload their tasks to edge servers, ensuring minimal system overhead. However, with the increasing task load, the system is not guaranteed to be stable.

II. System Model and Problem Statement

We consider a VEC system consisting of multiple vehicles and an edge server whose service range is a circle with a diameter and radius of 1000, which provides task offloading service for vehicles within its service range. In the VEC system, there is a group of vehicles numbered N={1,2,3...j...n}. As show in Fig.1, vehicle and the edge server are connected by wireless connection. In order not to waste the computing resources of the vehicle, we take the method of partial offloading, and denote the offloading ratio by x, where $x \in [0,1]$. Time is divided into a number of time slots, within each slot, the VEC server can complete the offloading decision for the corresponding vehicle task based on the task characteristics and computing resources.

A. Local Computing Mode

In the local computing model, there is no transmission delay of tasks, and local delay is mainly the execution time of tasks on the vehicle electronic control unit (ECU) [18]. Therefore, the delay and energy consumption of local computing can be expressed as:

$$T_{jv} = \frac{(1-x)\cdot M_j(t)\cdot s_v}{f_{vehicle}}, \tag{1}$$

$$E_{jv} = \frac{(1-x)\cdot M_j(t)\cdot s_v}{f_{vehicle}}\cdot p_v, \tag{2}$$

where $M_j(t)$ is the task size (in bits) generated by vehicle number j in time slot t, $f_{vehicle}$ denote the calculation speed of the ECU of the vehicle (cycles/second), and s_v is the number of CPU cycles required to compute each bit. p_v is the unit of power consumed by the vehicle computation, which is proportional to the square of $f_{vehicle}$ [18].

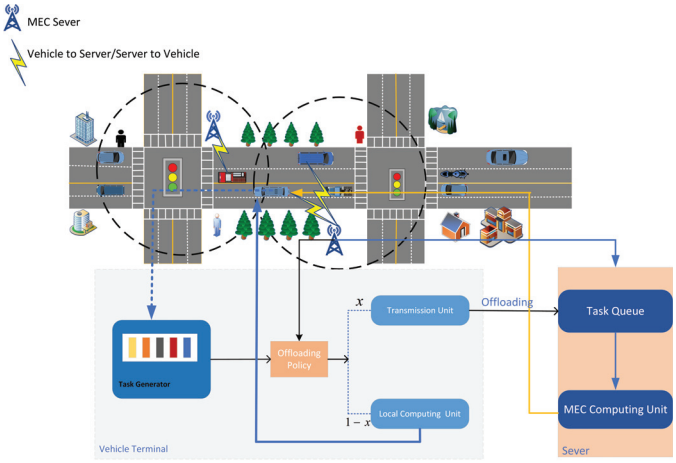

Fig. 1. Example of a VEC system

B. Edge Computing Mode

In the edge computing model, we consider the delay and energy consumption of the task transmission from the vehicle to the VEC server, as well as the delay and energy consumption of the task calculation by the VEC server. Because the calculation feedback time of the vehicle downloading the VEC server is very short, we ignore the delay and energy consumption in this process. The location of the VEC server is fixed in the VEC system [11] [19], and the position of vehicle j in slot t is expressed as $L_j(t) = [L_x, L_y, 0]^T$, the vehicle is traveling on a flat road at speed $v \in [0,20]$ in the same direction $\alpha \in [0,2\pi]$. Therefore, the position after the interval Δt is $L_j(t+\Delta t) = \left[L_x(t)+\nu(t)\cdot\Delta t\cdot\cos\alpha(t), L_y(t)+\nu(t)\cdot\Delta t\cdot\sin\alpha(t), 0\right]^T$. So the Euclidean distance between the vehicle and the VEC server at this time is $d_j(t) = \sqrt{\|L_j(t+\Delta t) - L_j(t)\|^2}$.

Therefore, the wireless transmission rate is shown in Eq.(3).

$$\mathrm{r}_j(t) = \omega\log_2\left(1+\frac{p\cdot\beta}{(\sigma^2+\varepsilon_j(t)P_{loss})\cdot d_j(t)}\right), \tag{3}$$

where ω denotes the bandwidth of the VEC system [20]. p is the uplink transmission power. σ^2 is the noise power. $\varepsilon_j(t) \in \{0,1\}$ is a flag for signal blocking. P_{loss} is the transmission loss power.

The task offload time and energy consumption can be represented by Eqs. (4) and (5).

$$T_{jr} = \frac{x\cdot M_j(t)}{\mathrm{r}_j(t)}, \tag{4}$$

$$E_{jr} = \frac{x\cdot M_j(t)\cdot s_\nu}{r_j(t)}p. \tag{5}$$

After the task arrives at the VEC server, it will enter the waiting queue and wait for the time to be executed as shown in Eq. (6).

$$T_{jw} = \frac{A_j(t)\cdot s_{vec}}{f_{vec}}, \tag{6}$$

$A_j(t)$ is the length of the waiting queue for time slot t.

The latency and energy consumption computed by the VEC server are expressed as shown in Eqs. (7) and (8).

$$T_{jvec} = \frac{x \cdot M_j(t) \cdot s_{vec}}{f_{vec}}, \tag{7}$$

$$E_{jvec} = \frac{x \cdot M_j(t) \cdot s_{vec}}{f_{vec}} \cdot p_{vec}, \tag{8}$$

where f_{vec} is the frequency of the VEC server and p_{vec} is the power.

C. Problem Statement

In this section, the long-term task offloading delay and energy sum minimization problem for all vehicles is transformed into a multi-constrained minimization problem [21], so we can formulate the problem as:

$$\text{(P1):D1} = \min \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} \sum_{j=1}^{N} C(t)[\lambda_1(t)T_j(t) + \lambda_2(t)E_j(t)] \tag{9}$$

subject to:

$$\boldsymbol{C1}: C(t) = \sum_{j=1}^{N} C(j) = 1,$$

$$\boldsymbol{C2}: \lambda_1(t) + \lambda_2(t) = 1, \forall \lambda_1(t), \lambda_2(t) \in [0,1],$$

$$\boldsymbol{C3}: T_j(t) = \max\{T_{jv}(t), T_{jr}(t) + T_{jw}(t) + T_{jvec}(t)\} \le T_{max}(t),$$

$$\boldsymbol{C4}: E_j(t) = E_{jv}(t) + E_{jr}(t) + E_{jvec}(t) \le E_{max}(t),$$

$$\boldsymbol{C5}: A_j(t) \le F_{vec}/n,$$

$$\boldsymbol{C6}: x_t \in [0,1],$$

where D1 is the system load at time slot t. Constraint $\boldsymbol{C1}$ implies that an offloading decision is made for one of the vehicles at each time slot. Constraint $\boldsymbol{C2}$ indicates that the sum of delay weight and energy consumption weight is 1. Constraint $\boldsymbol{C3}$ shows that the delay cannot exceed the maximum tolerated time of the task. F_{vec} is the memory of the edge server, constraint $\boldsymbol{C5}$ states that the length of the wait queue, for each vehicle cannot exceed the cache capacity allocated by the edge server.

One of the objectives of this paper is to optimize $C(t)$ and x_t to minimize the load D1 of the system.

III. Task Offload Optimization Algorithm

A. Probelm Conversion

Long-term timing constraints and optimization objectives pose a significant challenge to solve optimization problem P1 because it is challenging to obtain task information for future time slots, Lyapunov optimization method is a promising online optimization method, which only needs to ensure that the current time slot is optimal and can guarantee to obtain the optimal solution under long-term timing constraints [23]. We define N virtual task queues$\{A_j(t)\}_{j=1}^{N}$. Specifically, we set $A_j(0) = 0$ and update the queue as:

$$A_j(t+1) = \max(A_j(t) + D_j(t) - Q_j(t), 0), \tag{10}$$

where $D_j(t)$ is the newly created task size of the j vehicle in slot t, and $Q_j(t)$ is the processed task. Intuitively, when the newly generated task volume $D_j(t)$ is close to the processed task $Q_j(t)$ in time slot t, the task queue length in time slot $t+1$ is stable with that in time slot t.

Similarly, the energy consumption queue length can be dynamically expressed as:

$$Y_j(t+1) = \max(Y_j(t) + E_{jmax}(t) - E_{js}(t), 0), \tag{11}$$

where $E_{max}(t)$ in C4 is the energy consumption at the t th time frame, $E_{js}(t)$ is the energy supply.

To jointly control the data and energy queues, we define $\mathbf{Z}(t) = \left\{\{A_j(t)\}_{j=1}^{N}, \{Y_j(t)\}_{j=1}^{N}\right\}$ as the total queue backlog. Then, we introduce the Lyapunov function $L(\mathbf{Z}(t))$ and Lyapunov drift $\Delta L(\mathbf{Z}(t))$ as [17]

$$L(\mathbf{Z}(t)) = 0.5\left(\sum_{j=1}^{N} A_j(t)^2 + \sum_{j=1}^{N} Y_j(t)^2\right), \tag{12}$$

$$\Delta L(\mathbf{Z}(t)) = \mathbb{E}\{L(\mathbf{Z}(t+1)) - L(\mathbf{Z}(t)) | \mathbf{Z}(t)\}. \tag{13}$$

In order to stabilize the queue $\mathbf{Z}(t)$ while minimizing the system load, as shown in Eq.14, we use the drift plus penalty minimization method [22].

$$\Lambda(\mathbf{Z}(t)) \triangleq \Delta L(\mathbf{Z}(t)) + V \sum_{j=1}^{N} [\lambda_1(t)T_j(t) + \lambda_1(t)E_j(t)], \tag{14}$$

where $V > 0$ is an "importance" weight to scale the penalty.

By inference:

$\Lambda(\mathbf{Z}(t)) \le b_1 + b_2 + \sum_{j=1}^{N} A_j(t)\,[D_j(t) - Q_j(t)] + \sum_{j=1}^{N} Y_j(t)\,[E_{jmax}(t) - E_{js}(t)] + V\sum_{j=1}^{N}[\lambda_1(t)T_j(t) + \lambda_1(t)E_j(t)]$, where b_1 and b_2 are constant values. Similar proof processes can be referred to [17].

Therefore, the long-term optimization problem P1 is transformed into minimizing the drift plus penalty function in each time slot, and the objective function P1 is transformed into P2.

$$\text{(P2):D2=min}\mathbb{E}\{b_1 + b_2 + \sum_{j=1}^{N} A_j(t)\,[D_j(t) - Q_j(t)] + \sum_{j=1}^{N} Y_j(t)\,[E_{jmax}(t) - E_{js}(t)] + V\sum_{j=1}^{N}[\lambda_1(t)T_j(t) + \lambda_1(t)E_j(t)]\} \tag{15}$$

$$s.t. \boldsymbol{C1} - \boldsymbol{C6}.$$

Once P2 is determined for all time slots, the solution for P1 can be obtained. The drift plus penalty algorithm is able to ensure that the long-term constraint of stable queues is unchanged.

B. LYDDPG Based on Lyapunov Function

First, we will design state, action, and reward function, respectively.

- State: In the VEC system, the state space of reinforcement learning can be represented as:
  $$S_j(t) = (A_j(t), Y_j(t), L_j(t), M_j(t), \varepsilon_j(t)),$$
  $A_j(t)$ and $Y_j(t)$ bare the lengths of the task queue and the energy consumption queue, respectively, $L_j(t)$ is the position of the vehicle, $M_j(t)$ is the size of the new task, and $\varepsilon_j(t)$ is the channel blocking flag.

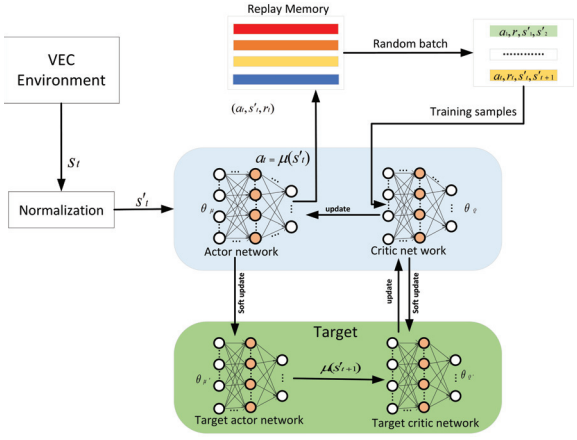

Fig. 2. LYDDPG network structure

- Action: According to the current state,the action a_t can be expressed as:
$$a_t = (k_t, x_t),$$
where k_t is the vehicle selected in time slot t and x_t is the task offloading ratio.
- Reward: Reward is a decisive factor in evaluating the performance of an algorithm. Let r_t = -D2.
- Policy: Task offloading based on reinforcement learning can be regarded as a Markov decision process(MDP). We adopt a deep deterministic policy, in each full Markov decision process, the strategy $\pi(a|s) = p(A_t = a|S_t = s)$ represents the percentages of choosing action a at current state s, until the end of the meet the termination conditions. where, $Q(s,a) = E(G|S_t = s, A_t = a)$ is the action state value function to evaluate the strategy.

Based on the above discussion, we propose an improved NLDDPG algorithm [16] called LYDDPG. As shown in Fig.2, adopt the Actor-Critic network, where the actor network formulates a deterministic action a_t based on the input state s_t. Critic network is used for policy evaluation and to improve algorithm stability. Denote θ_μ, θ_Q, $\theta_{\mu'}$, $\theta_{Q'}$ as parameters of the actor network, critic network, target actor network and target critic network.

Established a replay experience buffer R to train network parameters. In order to speed up the training inference, when the number of caches in the buffer reaches a batch, a batch of historical data is randomly taken from the buffer to train the network instead of waiting until the buffer is full before taking the experience replay session. To prevent gradient disappearance and gradient explosion during parameter learning, we introduce the LSTM network to prevent the gradient disappearance and explosion problem caused by long-term updates of the network.

The value range of state space is too extensive to reduce the training speed of neural network. To accelerate the convergence speed, we use a data preprocessing technique, namely state normalization strategy [16], normalizing state s_t to s'_t, which can make the input state value have a similar

Algorithm 1 The LYDDPG Training Algorithm

Input: 1. Initalize actor network $\pi(s, \theta_\mu)$ and critic network $Q(s, a|\theta_Q)$ with weights θ_μ and θ_Q.
2. Initalize target actor network μ' and target critic network Q' with weights $\theta'_\mu \leftarrow \theta_\mu$ and $\theta'_Q \leftarrow \theta_Q$.
3. The state s_t of the system and draw mini-batches of samples from the buffer.

Output: Make task offloading decisions and update the weights of the actor network.

for episode = 1 to M (Max episode) **do**
  Initalize a randomly process N for exploration
  Get the initial state s_0
  for each time slot = 1 to T **do**
    Normalize state s_t to s'_t
    Execute action $a_t = \mu(s_t|\theta_\mu) + N_t$ according to the current policy and exploration noise
    Calculate reward r_t, obtain a new state s_{t+1} and Normalize state s_{t+1} to s'_{t+1}
    if R is not full **then**
      Store transition $(a_t, r_t, s'_t, s'_{t+1})$ in experience replay buffer R
    else
      Randomly replace any other transition in R
    end if
    Sample a random minibatch of N transitions from the replay buffer R
    Update the actor policy using the sampled gradient:
$$\nabla\theta_\mu\mu|_{s_t} = \frac{1}{N}\sum_{j=1}^{N}\Big[\nabla_a Q(s,a|\theta_\mu)|_{s=s'_t,a=\mu(s'_t)|\theta_\mu} \nabla_{\theta_\mu}\mu(s|\theta_\mu)|_{s=s'_t}\Big]$$
    Q-value is calculated by the target critic network
    Set $y_t = r_t + \gamma Q(s'_{t+1}, \mu(s'_{t+1})|\theta_Q)$
    Update critic by minimizing the loss:
    $L(\theta_Q) = \frac{1}{N}\sum_{j=1}^{N}(y_t - Q(s'_t, a_t|\theta_Q))^2$
    Update the target networks:
    $\theta'_\mu = \tau\theta_\mu + (1-\tau)\theta'_\mu$
    $\theta'_Q = \tau\theta_Q(1-\tau)\theta'_Q$
  end for
end for

data range and reduce the difference between different state values. According to the normalized state s'_t, the actor network makes the offloading decision. At this time, the system state is normalized to s'_{t+1}, the state, action and reward are stored in the buffer. Cirtic network then fetches a bitch from the buffer for training and updates the weights of the actor network through gradient backpropagation. According to the state s'_{t+1}, the target actor network calculates the Q value of time slot $t+1$, and then updates the critic netwotk based on gradient backpropagation of target critic netwotk. Finally, the weights of the target actor network and target critic network are updated in the way of soft update. The pseudo-code of the training stage is described in Algorithm 1.

IV. Experimental results and analysis

In this section, the construction of the simulation environment and the simulation results are presented. We compare our algorithm with other algorithms in different settings. All the computations are evaluated on the TensorFlow platform with the 12-th generation Intel Core processor i7-12700 and 16 GB of the memory. The parameter settings are given in Table 1.

TABLE I
Parameter Settings

Parameter	Value
LSTM layers	2
Transmit power p	0.1
Batch size	64
Learning rate	6e-7
Soft update factor τ	0.01
Discount factor γ	0.99
$f_{vehicle}$	1.2GHZ
f_{vec}	48GHZ
Vehicular speed	[0-50]m/s
Task size	[0-256]MB
VEC server service radius	1000m
λ_1	0.5
λ_2	0.5
CPU cycles	1000 cycles/bit

The setting of hyperparameters is very important in the experiment, and different parameters will affect the stability and convergence of the results. The specific numerical parameter values of the VEC network used in the simulation are to verify the effectiveness of the algorithm. The parameter setting only needs to be set reasonably according to the actual environment. Fig.3 shows the impact of different transmit powers of vehicles

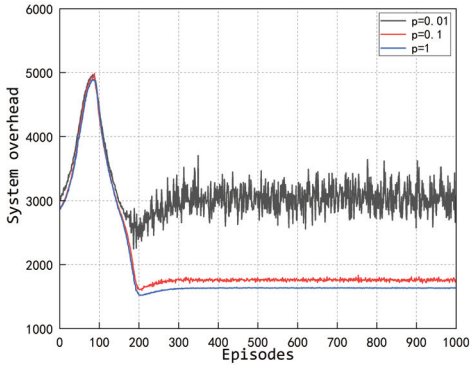

Fig. 3. Impact of transmit power p on system overhead

on the system overhead during task offloading. When p=0.01w, the task transmission time is too long, resulting in system instability and high load. When p=1, the system has the best stability and the minimum load, but only a little better than p=0.1w, and the power is ten times of it. Therefore, in order to ensure better convergence effect and energy saving, the transmit power is set to 0.1w during the task offloading.

In Fig.4, when the number of vehicles in the system is 10, compared with other methods, LYDDPG has the fastest convergence speed and the minimum system overhead. The DQN algorithm starts with low system overhead performance but has a great challenge in dealing with continuous action

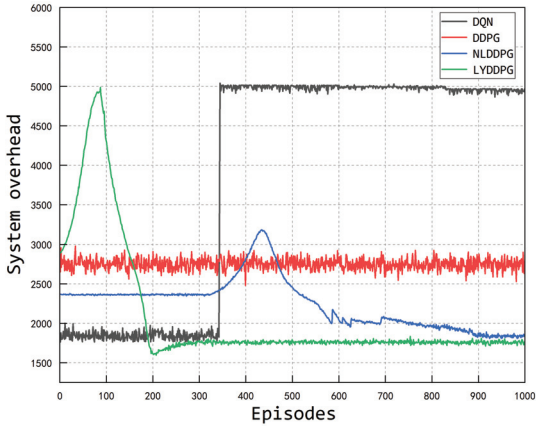

Fig. 4. System overhead for different algorithms

space problem. At the 345-th round, the system overhead increased dramatically as many tasks exceeded the neural network's processing capacity. The DDPG algorithm does not have any stability control, it is difficult to maintain stability, and has been bouncing back and forth between 2500-3000. The NLDDPG algorithm must wait until the buffer is full before starting training, and no system stability control is taken. It converges very slowly and sometimes cannot even maintain convergence.

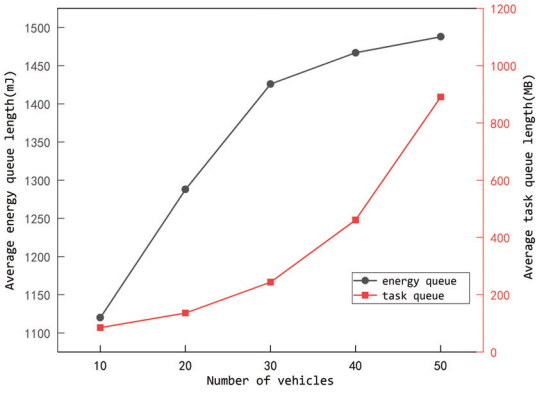

Fig. 5. The length of the task queue and energy queue

Fig.5 shows the average length of the task queue and the energy queue for the VEC system with different number of vehicles in the LYDDPG algorithm. When the number of vehicles in the system increases, the number of tasks offloaded to the edge server and the energy consumption also increases. However, we note that the increase in task queue and energy queue lengths is moderated.

Fig.6 compares the final load of the system for several algorithms with different numbers of vehicles. LYDDPG algorithm can still maintain the lowest load as the number of vehicles increases, and the load is not obvious with the increase of the number of vehicles. Due to the fact that the tasks of local computing algorithms are all completed by the vehicle itself, the load is not related to the number of vehicles in the VEC system, but only to the computational power of the vehicle itself. Random algorithms cause a large fluctuation

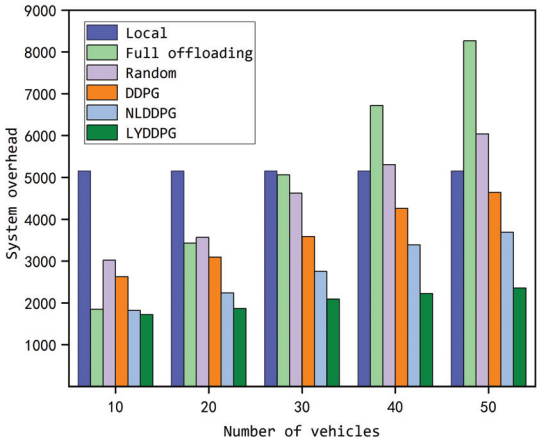

Fig. 6. Final system overhead of different vehicles

in system overhead due to the uncertainty of the offloading ratio in the process of each calculation. The full offloading algorithm assigns all tasks to the VEC server, which causes the computing resources of the VEC server to be strained, and does not make full use of the idle resources in the system. With the increase of the number of vehicles, the load is getting larger and larger. For DDPG and NLDDPG algorithms, as the number of vehicles increases, the system overhead also increases significantly.

V. CONCLUSION

This paper proposes Lyapunov-guided reinforcement learning for task offloading in VEC systems. We consider the impact of vehicle speed, position, signal interference, tasks of different sizes and conduct experiments under different numbers of vehicles to have the fastest convergence speed and the lowest system load while ensuring system stability. To improve the system's fault tolerance, make full use of the decentralized resources in the system, and prevent the system paralysis caused by centralized network failure. In the future, we consider a distributed offloading scheme where the offloading model is deployed on each vehicle instead of the edge server.

ACKNOWLEDGEMENT

This work is supported by the National Science Foundation of China (NSFC) under Grant 62262067, the Key Natural Science Foundation of Education Department of Anhui (KJ2021A0046).

REFERENCES

[1] A. Boualouache et al., "5G Vehicle-to-Everything at the Cross-Borders: Security Challenges and Opportunities," in IEEE Internet of Things Magazine, vol. 6, no. 1, pp. 114-119, March 2023, doi: 10.1109/IOTM.001.2200140.

[2] M. M. Saad, M. T. R. Khan, S. H. A. Shah and D. Kim, "Advancements in Vehicular Communication Technologies: C-V2X and NR-V2X Comparison," in IEEE Communications Magazine, vol. 59, no. 8, pp. 107-113, August 2021, doi: 10.1109/MCOM.101.2100119.

[3] B. B. Gupta, A. Gaurav, E. C. Marín and W. Alhalabi, "Novel Graph-Based Machine Learning Technique to Secure Smart Vehicles in Intelligent Transportation Systems," in IEEE Transactions on Intelligent Transportation Systems, vol. 24, no. 8, pp. 8483-8491, Aug. 2023, doi: 10.1109/TITS.2022.3174333.

[4] Q. Wu, H. Liu, R. Wang, P. Fan, Q. Fan and Z. Li, "Delay-Sensitive Task Offloading in the 802.11p-Based Vehicular Fog Computing Systems," in IEEE Internet of Things Journal, vol. 7, no. 1, pp. 773-785, Jan. 2020, doi: 10.1109/JIOT.2019.2953047.

[5] Q. Wu, H. Ge, H. Liu, Q. Fan, Z. Li and Z. Wang, "A Task Offloading Scheme in Vehicular Fog and Cloud Computing System," in IEEE Access, vol. 8, pp. 1173-1184, 2020, doi: 10.1109/ACCESS.2019.2961802.

[6] Luo J, Yin L, Hu J, et al. Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT[J]. Future Generation Computer Systems, 2019, 97: 50-60.

[7] Z. Sun, G. Sun, Y. Liu, J. Wang and D. Cao, "BARGAIN-MATCH: A Game Theoretical Approach for Resource Allocation and Task Offloading in Vehicular Edge Computing Networks," in IEEE Transactions on Mobile Computing, doi: 10.1109/TMC.2023.3239339.

[8] Islam A, Debnath A, Ghose M, et al. A survey on task offloading in multi-access edge computing[J]. Journal of Systems Architecture, 2021, 118: 102225.

[9] S. M. A. Kazmi et al., "A Novel Contract Theory-Based Incentive Mechanism for Cooperative Task-Offloading in Electrical Vehicular Networks," in IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 7, pp. 8380-8395, July 2022, doi: 10.1109/TITS.2021.3078913.

[10] J. Luo, X. Deng, H. Zhang and H. Qi, "Ultra-Low Latency Service Provision in Edge Computing," 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 2018, pp. 1-6, doi: 10.1109/ICC.2018.8422645.

[11] X. Huang, R. Yu, J. Kang and Y. Zhang, "Distributed Reputation Management for Secure and Efficient Vehicular Edge Computing and Networks," in IEEE Access, vol. 5, pp. 25408-25420, 2017, doi: 10.1109/ACCESS.2017.2769878.

[12] H. Guo, J. Zhang and J. Liu, "FiWi-Enhanced Vehicular Edge Computing Networks: Collaborative Task Offloading," in IEEE Vehicular Technology Magazine, vol. 14, no. 1, pp. 45-53, March 2019, doi: 10.1109/MVT.2018.2879537.

[13] Y. Han, Z. Zhao, J. Mo, C. Shu and G. Min, "Efficient Task Offloading with Dependency Guarantees in Ultra-Dense Edge Networks," 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9013142.

[14] Rahbari D, Nickray M. Low-latency and energy-efficient scheduling in fog-based IoT applications[J]. Turkish Journal of Electrical Engineering and Computer Sciences, 2019, 27(2): 1406-1427.

[15] Z. Gao, L. Yang and Y. Dai, "Fast Adaptive Task Offloading and Resource Allocation via Multiagent Reinforcement Learning in Heterogeneous Vehicular Fog Computing," in IEEE Internet of Things Journal, vol. 10, no. 8, pp. 6818-6835, 15 April15, 2023, doi: 10.1109/JIOT.2022.3228246.

[16] Hu X, Huang Y. Deep reinforcement learning based offloading decision algorithm for vehicular edge computing[J]. PeerJ Computer Science, 2022, 8: e1126.

[17] S. Bi, L. Huang, H. Wang and Y. -J. A. Zhang, "Lyapunov-Guided Deep Reinforcement Learning for Stable Online Computation Offloading in Mobile-Edge Computing Networks," in IEEE Transactions on Wireless Communications, vol. 20, no. 11, pp. 7519-7537, Nov. 2021, doi: 10.1109/TWC.2021.3085319.

[18] Long D, Wu Q, Fan Q, et al. A Power Allocation Scheme for MIMO-NOMA and D2D Vehicular Edge Computing Based on Decentralized DRL[J]. Sensors, 2023, 23(7): 3449.

[19] G. Ma, X. Wang, M. Hu, W. Ouyang, X. Chen and Y. Li, "DRL-Based Computation Offloading With Queue Stability for Vehicular-Cloud-Assisted Mobile Edge Computing Systems," in IEEE Transactions on Intelligent Vehicles, vol. 8, no. 4, pp. 2797-2809, April 2023, doi: 10.1109/TIV.2022.3225147.

[20] J. Chen, H. Xing, Z. Xiao, L. Xu and T. Tao, "A DRL Agent for Jointly Optimizing Computation Offloading and Resource Allocation in MEC," in IEEE Internet of Things Journal, vol. 8, no. 24, pp. 17508-17524, 15 Dec.15, 2021, doi: 10.1109/JIOT.2021.3081694.

[21] Yang J, Lin F, Saini D K, et al. Energy-efficient computation offloading and resource allocation in delay-constrained vehicular edge network[J]. International Journal of Communication Systems, 2022: e5335.

[22] Hu Y, Gong W, Zhou F. A Lyapunov-Optimized Dynamic Task Offloading Strategy for Satellite Edge Computing[J]. Applied Sciences, 2023, 13(7): 428.

[23] Neely M. Stochastic network optimization with application to communication and queueing systems[M]. Springer Nature, 2022.