Graph Neural Network Empowered Resource Allocation for Connected Autonomous Mobility

Eugen Šlapak*, Adam Petík*, Marcel Vološin*, Matúš Dopiriak*, Juraj Gazda*, Zdenek Becvar†
*Department of Computers and Informatics, Technical University of Košice
Košice, Slovakia
Email: eugen.slapak@tuke.sk
†Faculty of Electrical Engineering, Czech Technical University in Prague
Prague, Czech Republic
Email: zdenek.becvar@fel.cvut.cz

***Abstract*—Autonomous mobility and computations provided for passengers impose a high hardware and energy consumption related costs when deployed locally on connected autonomous vehicle (CAV). Distribution of resources for computation accross the edge of mobile network by means of multi-access edge computing (MEC) allows to reduce the cost of the CAVs. However, the decision on computation offloading and allocation of resources for computing is itself a computationally complex task. Existing works typically do not fully exploit the potential of machine learning by combining novel advances in deep reinforcement learning (DRL) and graph neural networks (GNNs) that are suited for graph structure of the MEC. We propose a novel framework combining GNNs with deep deterministic policy gradient (DDPG) variant of DRL. The proposed concept is tested in environment with simulated gNodeBs, CAVs and execution of actions that simultaneously trade off uplink and processing resources and control the soft deadline buffer. In scenario with one base station and 12 CAVs our approach outperforms commonly used multilayer perceptron DDPG by 59% in terms of failed task ratio metric. Additionally, in scenario with 3 base stations and 25 CAVs, the proposal reaches over 33% for the same metric over round robin (RR) distribution.**

***Index Terms*—autonomous mobility, multi-access edge computing, resource allocation, graph neural network, DDPG**

I. Introduction

Autonomous mobility can transform society by improving transportation, urban planning, and environment. According to the Deloitte report [1], most experts expect autonomous vehicles to be widely adopted after 2030. Dubey et al. [2] predict that they will capture 50% of the market in 18 years and 80% in 31 years. Autonomous vehicles need a high computing power for self-driving and other functions. Putting such resources on connected autonomous vehicles (CAVs) raises costs and energy consumption, and reduces the vehicle's driving range.

Higher computing power enhances self-driving cars' accuracy, reliability, latency and safety. Using remote servers, vehicles can achieve otherwise impossible improvements. Multi-access edge computing (MEC) is a good solution for CAV mobility, which needs fast computing and communication. MEC does computation and data processing at the network edge, near the data source. However, MEC also poses challenges for resource allocation, as the CAVs have to allocate the resources effectively among them and the MEC servers.

Energy-efficient allocation for computing and caching in cellular networks is the topic of study in [3]. The authors use deep deterministic policy gradient (DDPG) to solve the problem with RL and show over 15% energy saving and timely task processing. In [4], deep reinforcement learning (DRL) and TD3 form the basis for computation offloading and resource allocation algorithm, to optimize MEC offloading and allocation for internet of vehicles (IoV) with different computational tasks. The algorithm achieves better delay, cost, speed, and stability. The research in [5] also uses DRL with TD3, but tackles optimization of latency and power by dynamically allocating MEC resources. The action decides whether to send the task to MEC and CPU and communication resource use. In [6] authors use multi-agent deep deterministic policy gradient (MADDPG) model to decide MEC or gNB offloading and allocation. The agent influences the neighboring gNBs' actions.

Graph neural network (GNN) and DRL are used in [7] and [8] to model the network as a graph and use GNN to get node and link features. In [7] this approach minimizes latency by choosing MEC or local computing, while in [8] it assigns spectrum for D2D communication in vehicular networks. Works in [9] and [10] use supervised and semi-supervised GNN learning respectively. In [9] GNN is trained with samples from a suboptimal *constraint cross-entropy* method for resource allocation, while in [10] some devices have fixed gNB connections to serve as supervision labels, to train the model for power and gNB selection in densely connected networks

Review in [11] surveys GNNs in wireless networks and finds many works on power, channel, or spectrum allocation, but few on computation and communication allocation. RL learns optimal policies from trial-and-error, while GNNs handle well the graph data and dependencies. MEC networks have inherent graph structure and GNNs can represent and extract their features. But most works on MEC resource allocation use RL or GNNs separately, or for different aspects.

The main contributions of this paper are summarized as follows:

1) We propose a novel GNN and DDPG method for computing resource allocation in VEC.

2) We describe novel ways to combine GNN and DRL for control tasks with graph states. Such combination can be also applied to problems beyond resource allocation.
3) We demonstrate that our method improves performance metrics compared to related works for various scenarios with different simulation settings, such as number of gNBs, CAVs, size of tasks, and deadline constraints.

II. System model

This paper aims to compute tasks within the designated deadline $t_{\mathrm{deadline},i}$, indicated for the specific task indexed as i. When allocating resources, $t_{\mathrm{deadline},i}$ is not directly utilized; instead, the adjustment is made using the soft task deadline $t_{\mathrm{soft\_deadline},i}$ to account for environmental unpredictability. $t_{\mathrm{soft\_deadline},i}$ is calculated as follows:

$$t_{\mathrm{soft\_deadline},i} = t_{\mathrm{deadline},i}\ (1-\epsilon_i), \tag{1}$$

where $\epsilon_i \in \langle 0,1\rangle$ is the $t_{\mathrm{soft\_deadline},i}$ contraction factor that shortens the $t_{\mathrm{soft\_deadline},i}$ relative to $t_{\mathrm{deadline},i}$.

The total time for transmitting the i-th task to the server, computing it, and sending the response back to CAV is denoted as $t_{\mathrm{total},i}$. If the server response size is small enough to be negligible, $t_{\mathrm{total},i}$ can be calculated as:

$$t_{\mathrm{total},i} = t_{\mathrm{uplink},i} + t_{\mathrm{process},i}, \tag{2}$$

where $t_{\mathrm{uplink},i}$ is the uplink time and $t_{\mathrm{process},i}$ is the processing time for the i-th task.

Resource distribution is determined according to partial soft deadlines $t_{\mathrm{soft\_uplink},i}$ for uplink and $t_{\mathrm{soft\_process},i}$ for processing, from $t_{\mathrm{soft\_deadline},i}$ as:

$$\begin{aligned} t_{\mathrm{soft\_uplink},i} &= \omega_i\ t_{\mathrm{soft\_deadline},i}, \\ t_{\mathrm{soft\_process},i} &= (1-\omega_i)\ t_{\mathrm{soft\_deadline},i}, \end{aligned} \tag{3}$$

where ω_i is a parameter that specifies the resource split ratio. Once specific ω_i is chosen, required uplink throughput and computational resources that will satisfy both soft deadlines can be calculated. The following formula is used to calculate the uplink data rate between CAV u and gNB s at time t:

$$R_{u,s}(t) = \beta_u(t)\rho_u(t)f_{\mathrm{s}}(t)n_{\mathrm{RB},u,s}(t), \tag{4}$$

where β_u is the number of bits per symbol, ρ_u is the code rate, f_{s} is the modulation rate, and $n_{\mathrm{RB},u,s}(t)$ expresses the number of resource blocks allocated by the s-th gNB for the u-th CAV. The values of $\beta_u(t)$ and $\rho_u(t)$ are determined from the measured signal-to-noise (SINR) value at CAV coordinates, with mapping table available in [12]. f_{s} can be calculated from number of resource elements available in a single resource block.

Equation (4) can be modified to get the required number of resource blocks $n_{\mathrm{RB},u,s}$ to achieve the desired data rate:

$$n_{\mathrm{RB},u,s}(t) = \left\lceil \frac{R_{u,s}(t)}{\beta_u(t)\rho_u(t)f_s(t)} \right\rceil. \tag{5}$$

The time $t_{\mathrm{soft\_uplink},i}$ is known and it can be used to calculate the desired data rate as follows:

$$R_{u,s}(t) = D_i/t_{\mathrm{soft\_uplink},i}(t), \tag{6}$$

where D_i is the size of the task in megabits at upload.

The CPU capacity of MEC resources will be expressed in instructions per second (IPS) unit. The required CPU resources for a task that should be processed by time $t_{\mathrm{soft\_process},i}$ are calculated as follows:

$$I_{\mathrm{CPU},u,s} = I_i/t_{\mathrm{soft\_process},i}, \tag{7}$$

where I_i is the number of instructions needed to compute the i-th task on the CPU.

Our simulations use two metrics that are also used to measure the service quality and for our objective function on which models are trained. The first metric represents ratio of failed tasks, i.e., tasks not completed within the deadline, and is defined as

$$M_{\mathrm{FT}} = \frac{N_{FT}}{N_{GT}}, \tag{8}$$

where N_{FT} is the total number of failed tasks that exceeded the allowed latency $t_{\mathrm{deadline},i}$, thus do not fulfill the constraint $t_{\mathrm{total},i} \le t_{\mathrm{deadline},i}$ and N_{GT} is a total number of generated tasks.

The second metric, denoted as M_{L}, is an average latency, expressing the average ratio of $t_{\mathrm{total},i}$ to $t_{\mathrm{deadline},i}$ for all tasks. Now the multicriterial objective function that takes into account both M_{FT} and M_{L} can be defined as:

$$P1: \min_{N_{\mathrm{CAV}},D_i,I_i} \Big(M_{\mathrm{FT}}, M_{\mathrm{L}}\Big) \tag{9}$$

s.t.:

$$\begin{aligned} &C1.1: \sum_u n_{\mathrm{RB},u,s}(t)\ \le N_{\mathrm{RB},s}, \\ &C1.2: \sum_u I_{\mathrm{CPU},u,s}(t)\ \le I_{\mathrm{CPU},s}, \\ &C1.3: 0<\omega_i<1 \\ &C1.4: 0<\epsilon_i<1, \end{aligned}$$

where $C1.1$ defines the limit of assigned resource blocks in time t, so that it does not exceed the maximal number of RBs of s-th gNB, $C1.2$ defines limitation of assigned CPU resources in time t, so that they will not exceed the total amount of CPU resources of s-th gNB, $C1.3$ limits the resource distribution ratio ω_i for i-th task so that resource assignment stays in $\langle 0\%, 100\%\rangle$ range and $C1.4$ indirectly limits the soft deadline ratio $t_{\mathrm{soft\_deadline},i}$.

III. Proposed approach

Our approach uses transformation of a network state graph by graph convolutional network (GCN) [13] layer to obtain abstract features for the DDPG agent. GCN layer aggregates attributes of vertex and its neighbors. If all vertices are interconnected, like all CAVs connected to the same gNB, GCN gives identical vectors for all of them.

The $\mathbf{x}_u$ feature vector of the u-th CAV has the following form:

$$\mathbf{x}_u = (sinr_u, sinr_{\mathrm{old},u}, n_{\mathrm{RB},u,s}, I_{\mathrm{CPU},u,s}), \tag{10}$$

where $sinr_u$ is the current measured value of SINR of the u-th CAV, $sinr_{\mathrm{old},u}$ is the SINR value measured in previous time step, $n_{\mathrm{RB},u,s}$ and $I_{\mathrm{CPU},u,s}$ are the current reserved number of resource blocks and the current reserved CPU computation power, respectively.

The reward function is defined as follows:

$$r_u(t) = \begin{cases} N_{CT,s}(t), & \text{if } N_{FT,s} > 0 \\ N_{CT,s}(t)\ (1 + L_s(t)), & \text{else} \end{cases} \tag{11}$$

where $N_{CT,s}(t)$ is the number of all processed tasks of CAVs connected to s-th gNB in time interval t and $L_s(t)$ represents the average ratio of time needed to compute the task to the overall time limit for task processing.

Now that our metrics and reward calculation that guide the model training are introduced, our DDPG implementation can be described. DDPG algorithm consists of replay buffer, actor, critic, and their target networks. Replay buffer saves (s,a,r,s',d) tuples for state, action, reward, next state, and binary d, that equals True if s' is terminal. gNBs constantly record and store these tuples into replay buffer.

We also use GCN layer as feature extractor for actor and critic. It converts graph state of gNB and CAVs to feature vector. First, GCN generates feature vectors for all relevant network nodes, denoted as $\mathbf{h_1} \dots \mathbf{h_n}$. We use the node vectors to make state vector $\mathbf{h}$ by aggregation and concatenation, $\mathbf{h}$ is be passed to densely connected layers of the deep neural network (DNN) to finally generate either action in case of actor network, or state-action evaluation, in case of critic network.

For the actor, aggregated node vectors are extended as

$$\mathbf{h} = \mathrm{concat}\left(\mathrm{aggr}\left(\mathbf{h_1}, \dots, \mathbf{h_n}\right), \mathbf{x}\right), \tag{12}$$

where *concat* denotes the vector concatenation function, *aggr* denotes the vector aggregation function and $\mathbf{x}$ is a feature vector of specific CAV. Concatenation of $\mathbf{x}$ ensures that state vector for each CAV requiring control by actor will be unique and CAV-specific information will be reintroduced.

Critic obtains $\mathbf{h}$ in the same manner, but since it evaluates state-action pair, it adds action tuple to the concatenation as

$$\mathbf{h} = \mathrm{concat}\left(\mathrm{aggr}\left(\mathbf{h_1}, \dots, \mathbf{h_n}\right), \mathbf{x}, a\right). \tag{13}$$

Our GNN_CAV approach lets each CAV to execute ω_i separately. It thus works for any number of CAVs. We use one critic for all gNBs, but each gNB has its own actor. Actor and critic weights are updated in the cloud and shared with all gNBs.

The DDPG agent trains on a randomly sampled batch B from memory buffer. The target actor and target critic models respectively generate and evaluate actions for new states. The state-action pair value determined by target critic is used to compute the target value. With access to state of gNB and feature vectors of CAVs and features, the agent can allocate MEC resources individually for each CAV and should perform more optimally over uniform distribution.

TABLE I: Simulation settings

Parameter	Value
Δt_{T}	0.5 s
$t_{\mathrm{deadline},i}$	0.1 s
N_{gNB}	1, 2, 3, 4, 5
$N_{\mathrm{RB},s}$ (number of RBs per gNB)	1000
v (CAV speed)	$\langle 35, 50 \rangle$ km/h
D_i	0.1 MB
I_i	10 MI
$I_{\mathrm{CPU},s}$	$3 * 10^9$ IPS
P_{t}	0.01 W
f_{t}	$2 * 10^9$ Hz

IV. Simulation results

We simulate CAV mobility on a 200 × 200 m grid of streets. CAVs pick random destinations after reaching one. The model contains N_{gNB} gNBs with MEC servers and N_{CAV} CAVs that send tasks to MEC network. We only consider task computation on gNB connected to CAV, without possibility of computing them locally on CAV. Simulation steps generate tasks, collect results, and update agent rewards.

Throughout the simulation scope the performance characteristics of the proposed method denoted as **GNN_CAV** performing resource allocation action at the level of CAVs using GNN are provided. Monte Carlo simulations are used with settings in Table I. Each step is 0.5 s and generates five tasks. gNB number varies up to five. We adjust gNB parameters like N_{gNB}, $N_{\mathrm{RB},s}$, and task parameters for smaller environments and fewer CAVs so that in certain situations there are still insufficient MEC resources for all CAVs, to test different network loads.

A. Baselines

We compare our approach with other resource allocation methods, including DL ones.

Round robin (RR) : RR distributes MEC resources equally among CAVs on the same gNB.

NN_CAV: NN_CAV uses DDPG without GNN and allocates resources at CAV level. It is similar to approach in [14]. The state has features of all CAVs and the action is separate for each CAV. It works only for fixed gNB and CAV numbers, therefore we use it only in some plotted scenarios.

GNN_gNB: GNN_gNB uses DDPG with GNN at gNB level. It gives the same VEC resources to all CAVs on the same gNB.

B. Single gNB in the environment

This section describes the simulation results of a simplified scenario involving a single gNB. Table II illustrates the proportion of failed tasks M_{FT} and latency metric M_{L} across various scenarios with different numbers of CAVs, as a comprehensive performance comparison of the evaluated methods. Agents act with action space $a = (\omega_i, \epsilon_i)$, influencing resource allocation (ω_i) and latency (ϵ_i). We study the average failed tasks (M_{FT}) and latency (M_{L}). The setup parameter $t_{\mathrm{deadline},i}$ sets the maximum delay of 0.09 seconds, plus 0.01 seconds as a buffer accounting for unpredictability of future resource load levels.

With 7 CAVs and enough resources, all methods have M_{FT} close to 0. With 12 CAVs and more failed tasks due to higher resource load, RR performs notably worse than NN_CAV, while GNN_CAV performs better than NN_CAV, but worse than GNN_gNB. The M_{FT} value of 0.043 for GNN_CAV in this scenario is nearly double that of GNN_gNB. With 15 CAVs and higher load, GNN_CAV has the lowest M_{FT} of 0.213, followed by GNN_gNB with M_{FT} of 0.228, while RR falls behind all other approaches.

In conclusion, scenarios featuring more CAVs require more advanced approaches for maintaining network reliability. Among the tested methods, proposed method GNN_CAV emerges as the most effective in terms of M_{FT}, providing the best performance results.

GNN_gNB algorithm achieves the average latency of 0.052s in the scenario with 7 CAVs with other methods achieving very similar values. In setting with 12 and 15 CAVs M_{L} of GNN_gNB, GNN_CAV and NN_CAV approach to a maximum possible value of 0.09s. The findings manifest that RR attains the most favorable mean latency performance compared to the alternative tested methodologies but at the expense of worse failed task ratio for settings with higher N_{CAV}.

TABLE II: (M_{FT}) and average (M_{L}) for action $a = (\omega_i, \epsilon_i)$

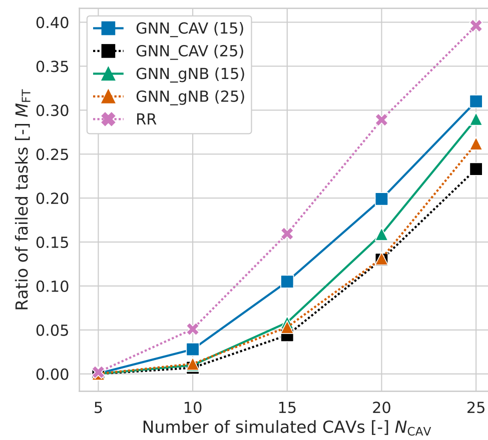
N_{CAV}	RR		GNN_gNB		GNN_CAV		NN_CAV	
	M_{FT}	M_{L}	M_{FT}	M_{L}	M_{FT}	M_{L}	M_{FT}	M_{L}
7	0.000	0.048	0.003	0.052	0.004	0.049	0.002	0.051
12	0.173	0.073	0.028	0.090	0.043	0.084	0.069	0.089
15	0.390	0.082	0.228	0.090	0.213	0.090	0.271	0.090

C. Multiple gNBs in the environment

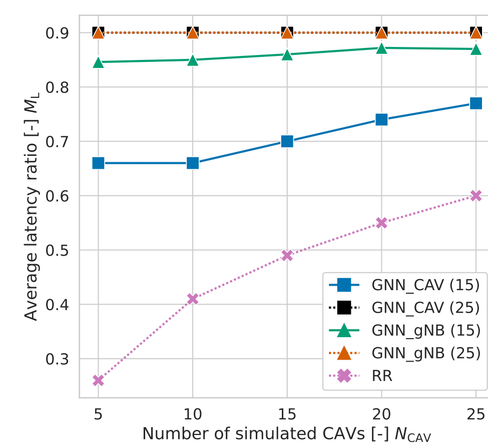
Real-world scenarios often include multiple gNBs. Thus, we have trained and tested agents in multi-gNB scenarios to evaluate the impact of environment properties on GNN's training. Agents were trained with 3 gNBs and 15 or 25 CAVs, and tested with varying numbers of gNBs and CAVs. NN_CAV was not used as it cannot handle different numbers of CAVs.

Simulation results from scenarios with agents executing actions $a = (\omega, \epsilon)$ for varying number of CAVs in the environment are shown in Fig. 1. It displays the impact of the number of CAVs in the environment with 3gNBs on the M_{FT} and M_{L} metric of RR algorithm and two variants of each GNN_CAV and GNN_gNB (variants were trained in the environment with number of CAVs fixed to 15 and 25 respectively, shown in the brackets). It is clear, that in terms of M_{FT}, GNN-based approaches have significant advantage over the RR in each setting. GNN_CAV(25) has the lowest average M_{FT} across the majority of tested scenarios. Experiments also show, that agent variants trained with more CAVs present show improved relative performance compared to RR.

In general, more CAVs also result in higher average latency. RR algorithm significantly outperforms its counterparts in terms of M_{L} metric but at the expense of a less favorable M_{FT}. Similarly, the GNN_CAV(15) demonstrates lower M_{L} than other GNN-based methods thus is unable to capitalize on the $soft\_deadline$ to redistribute resources in more optimal manner, a capability that both GNN_CAV(25) and GNN_gNB(25) leverage effectively with the average latency ratio of $M_{\mathrm{L}} = 0.9$.



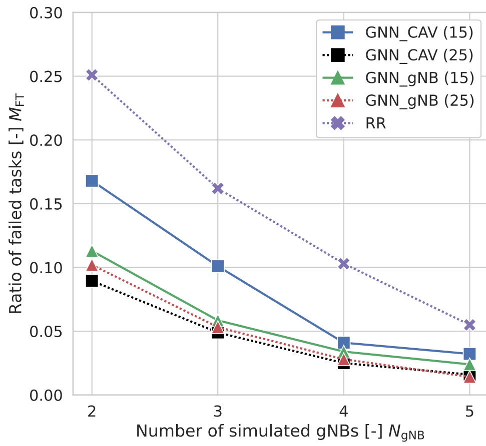
(a)



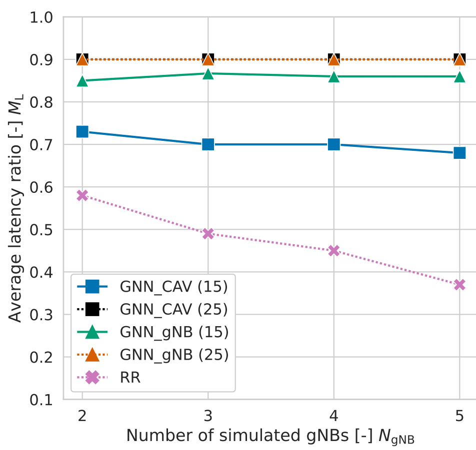
(b)

Fig. 1: Effects of action $a = (\omega, \epsilon)$ with 3 gNBs and different number of CAVs: (a) M_{FT} dependence on CAV number, (b) M_{L} dependence on CAV number

Fig. 2 shows the impact of number of gNBs on M_{FT} and M_{L} in settings with agents executing resource allocation action and soft deadline determination action $a = (\omega, \epsilon)$ and with CAV number fixed to 15. In general, rising number of gNBs present in environment decreases both the M_{FT} and the M_{L}. GNN_CAV(25) shows superior results in terms of average ratio of failed tasks. As for the average latency ratio M_{L}, RR and CNN_CAV(15) outperform other approaches at the expense of M_{FT}. Again, both GNN_CAV(25) and GNN_gNB(25) have the average latency ratio $M_{\mathrm{L}} = 0.9$ which equals the value of $soft\_deadline$.



(a)



(b)

Fig. 2: Effects of action $a = (\omega, \epsilon)$ with 15 CAVs and different number of gNBs: (a) M_{FT} dependence on gNB number, (b) average M_{L} dependence on gNB number

V. Conclusion

This work proposes novel GNN algorithms for VEC resource allocation. Their performance is comprehensively compared with RR and NN_CAV methods using extensive Monte Carlo simulations. The findings demonstrate the superiority of both proposed GNN-based algorithms. This translates to savings in operational cost, improved QoS and M_{FT} task metric. More CAVs during the training improve GNN results relative to other approaches.

Acknowledgment

This work was supported by the by the Slovak Research and Development Agency under Grants APVV SK-CZ-RD-21-0028 and APVV-21-0318, by Ministry of Education, Science, Research and Sport of the Slovak Republic, and the Slovak Academy of Sciences under Grant VEGA 1/0685/23, and by the Ministry of Education, Youth and Sports, Czech Republic as a part of the project no. LUASK22064.

References

[1] H. Proff, T. Pottebaum, and P. Wolf, "Autonomous driving - moonshot project with quantum leap from hardware to software & ai focus," Deloitte, , 2019.

[2] S. Dubey, I. Sharma, S. Mishra, O. Cats, and P. Bansal, "A general framework to forecast the adoption of novel products: A case of autonomous vehicles," *Transportation research part B: methodological*, vol. 165, pp. 63–95, 2022.

[3] X. Kong, G. Duan, M. Hou, G. Shen, H. Wang, X. Yan, and M. Collotta, "Deep reinforcement learning-based energy-efficient edge computing for internet of vehicles," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 9, pp. 6308–6316, 2022.

[4] J. Huang, J. Wan, B. Lv, Q. Ye, and Y. Chen, "Joint computation offloading and resource allocation for edge-cloud collaboration in internet of vehicles via deep reinforcement learning," *IEEE Systems Journal*, 2023.

[5] L. Yao, X. Xu, M. Bilal, and H. Wang, "Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–9, 2022.

[6] Y. Lu, D. Han, X. Wang, and Q. Gao, "Distributed task offloading for large-scale vec systems: A multi-agent deep reinforcement learning method," in *2022 14th International Conference on Communication Software and Networks (ICCSN)*, 2022, pp. 161–165.

[7] Z. Sun, Y. Mo, and C. Yu, "Graph reinforcement learning based task offloading for multi-access edge computing," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[8] Z. He, L. Wang, H. Ye, G. Y. Li, and B.-H. F. Juang, "Resource allocation based on graph neural networks in vehicular communications," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–5.

[9] T. Chen, X. Zhang, M. You, G. Zheng, and S. Lambotharan, "A gnn-based supervised learning framework for resource allocation in wireless iot networks," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 1712–1724, 2022.

[10] X. Zhang, Z. Zhang, and L. Yang, "Joint user association and power allocation in heterogeneous ultra dense network via semi-supervised representation learning," *CoRR*, vol. abs/2103.15367, 2021. [Online]. Available: https://arxiv.org/abs/2103.15367

[11] S. He, S. Xiong, Y. Ou, J. Zhang, J. Wang, Y. Huang, and Y. Zhang, "An overview on the application of graph neural networks in wireless networks," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 2547–2565, 2021.

[12] A. Chiumento, M. Bennis, C. Desset, L. van der perre, and S. Pollin, "Adaptive csi and feedback estimation in lte and beyond: a gaussian process regression approach," *EURASIP Journal on Wireless Communications and Networking*, vol. 2015, 12 2015.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[14] L. Yao, X. Xu, M. Bilal, and H. Wang, "Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, 2022.