TransEdge: Task Offloading With GNN and DRL in Edge-Computing-Enabled Transportation Systems

Aikun Xu, Zhigang Hu, Xi Li, Rongti Tian, Xinyu Zhang, Bolei Chen, Hui Xiao, Hao Zheng, *Graduate Student Member, IEEE*, Xianting Feng, Meiguang Zheng, Ping Zhong, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—In recent years, since edge computing has improved the performance of transportation systems, research on edge-computing-enabled transportation systems has received widespread attention. However, most previous studies overlooked that task requests in transportation systems are unevenly distributed in time and space, which easily causes the overloading of edge servers, resulting in high response latency. To this end, we present a novel task offloading scheme based on graph neural network (GNN) and deep reinforcement learning (DRL) in edge-computing-enabled transportation systems (TransEdge). Specifically, we first propose an adaptive node placement algorithm to assign Internet of Things sensors to appropriate edge servers, thereby minimizing transmission latency. Then, an improved DRL scheme based on GNN is designed to capture the spatial features between sensors, aiming to improve the accuracy of task offloading decisions. Finally, we introduce a task forwarding strategy based on the greedy algorithm to achieve collaborative task offloading between different edge servers and overcome the system instability caused by a sudden surge in task requests. We conduct extensive experiments on two real-world traffic data sets. The results show that TransEdge reduces the response latency by at least 3.7% compared to four baselines while achieving a success rate of 99%.**

***Index Terms*—Deep reinforcement learning (DRL), edge computing, graph neural network (GNN), task offloading, transportation systems.**

I. Introduction

THE EMERGENCE of the Internet of Things (IoT) technology has promoted the transformation of traditional vehicles to connected vehicles, thereby enhancing the development of conventional transportation systems to intelligent transportation systems [1]. In the above intelligent transportation system, we need to process a large amount of data (e.g., navigation data, video data, audio data, etc.) generated by vehicles with the help of roadside sensors (i.e., IoT sensors), which aims to provide users with road safety and driving comfort [2]. This scenario poses higher demands on the computational capability, processing latency, and memory consumption of the sensor hardware [3]. However, the resources are very limited in sensors, and processing raw data directly with sensors may potentially lead to high latency [4]. To address this problem, most of the previous works directly send the data collected by the sensor to a remote central cloud for analysis, subsequently receiving the processed results from the cloud. This leverages sufficient resources from the cloud to make up for the lack of the sensors' own resources. Nevertheless, the sensor transmits a large amount of data from the network's edge to the central cloud, which burdens the network transmission bandwidth, leading to intolerant transmission latency (TL). It can be seen that cloud computing struggles to support real-time computation for the vast amount of data in transportation systems [5].

Fortunately, edge computing is an outstanding computing paradigm that promises to provide low-latency services in transportation systems [6], [7]. It sinks resources from the central cloud to edge servers closer to users, while applying task offloading to transmit data (i.e., tasks) from IoT sensors to nearby edge servers to provide reliable, efficient, and low-latency services [8], [9]. It is evident that edge computing harbors immense potential for saving TL, alleviating network bandwidth pressure, and reducing energy consumption [10]. Nevertheless, without a proper task offloading method among edge servers (e.g., collaborative task offloading [11]), it is difficult to fully exploit the performance advantages of edge computing, and it may lead to excessively high response latency and energy consumption due to edge server overload [12].

Researchers have made many efforts to solve the above problems. For example, Chen et al. [13] defined task software cache update and computation offloading as a joint optimization problem. For the computation offloading subproblem, a decentralized algorithm is proposed to find the Nash equilibrium solution. Next, a method based on double deep Q-network is introduced to solve the task software cache update subproblem. Although these schemes achieve good performance, they are difficult to scale to scenarios involving

Manuscript received 30 May 2024; revised 8 August 2024; accepted 12 August 2024. Date of publication 15 August 2024; date of current version 20 November 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62172442 and Grant 62272489; in part by the National Science Foundation of Hunan Province under Grant 2022JJ30760; and in part by the Fundamental Research Funds for the Central Universities of Central South University under Grant 512340030. *(Corresponding authors: Zhigang Hu; Xi Li.)*

Aikun Xu, Zhigang Hu, Xi Li, Rongti Tian, Xinyu Zhang, Bolei Chen, Hui Xiao, Hao Zheng, Xianting Feng, Meiguang Zheng, and Ping Zhong are with the School of Computer Science and Engineering, Central South University, Changsha 410073, China (e-mail: aikunxu@csu.edu.cn; zghu@csu.edu.cn; lixi@csu.edu.cn; tianrongti@163.com; zhangxinyu11014@163.com; boleichen@csu.edu.cn; huixiao@csu.edu.cn; zhenghao@csu.edu.cn; xiantingfeng@csu.edu.cn; zhengmeiguang@csu.edu.cn; ping.zhong@csu.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/JIOT.2024.3443866

multiple edge servers. To this end, Liu et al. [14] jointly optimized the computation offloading and resource allocation problems in multiuser edge computing networks by taking into account the transmission power level, subchannel, and wireless access technologies. Then, the problem is modeled as a Markov game process, and an independent learners-based multiagent Q-learning algorithm is proposed to solve it, aiming to improve the performance of the system. Chen et al. [15] first transformed the communication and computing resource management strategy of each edge server into a digital twin migration problem. Next, the problem is modeled as a decentralized partially observable Markov decision process. Finally, a novel agent-contribution-enabled multiagent reinforcement learning algorithm is introduced to solve it. Nevertheless, transportation scenarios usually contain many IoT sensors. If corresponding deep learning models are deployed on each sensor, frequent training or inference of these models will cause a large overhead. Last but not least, the uneven temporal and spatial distribution of task requests in transportation systems [16], leads to the overload of edge servers at certain times. In this case, even the ideal task offloading strategy fails to ensure the performance of edge-computing-enabled transportation systems.

To solve this challenge, scholars have proposed some novel solutions. For example, Xu et al. [3] first used the graph neural network (GNN, which is a deep learning model that aims to extract as much potential representation information in the "graph" as possible [17]) to predict traffic flow in the transportation system. Then, the authors estimated the result of resource allocation according to the flow value. Finally, they implemented a task offloading scheme between edge servers based on deep reinforcement learning (DRL, which combines deep learning and reinforcement learning to solve problems that require a series of decisions [18]). Liu et al. [19] separately considered peak and off-peak periods in the task offloading decision problem. For off-peak periods, task requests are directly offloaded to the edge server, which is modeled as the integer programming problem (IP). The above IP problem is addressed using the simulated annealing genetic algorithm (SAGA). For peak periods, task requests have the option to be handled locally or offloaded to the edge server, with solutions provided by deep Q network (DQN). However, these approaches have the following shortcomings: 1) they overlook the significance of preplacement of sensors, resulting in increased TL; 2) the reinforcement learning schemes used fail to capture spatial features of task requests, diminishing the rationality of task offloading decisions; and 3) they lack an effective task forwarding strategy, making it challenging to achieve reasonable collaborative task offloading when facing a sudden increase in task requests.

To tackle the challenges above, we provide a novel task offloading scheme with GNN and DRL in edge-computing-enabled transportation system, namely, TransEdge. Specifically, we first design an adaptive genetic algorithm (AGA) to achieve reasonable placement of IoT sensors and reduce TL. Next, an improved reinforcement learning scheme based on GNN is proposed, which empowers reinforcement learning to capture the spatial correlation relationship between tasks and enhance the accuracy of task offloading decisions. Subsequently, a task forwarding strategy based on greedy algorithm (which always makes the most advantageous choice in the current state) is introduced to alleviate the edge server overload problem. Finally, the simulation results show the advantages of TransEdge. The main contributions of our work are as follows.

1) We propose TransEdge, a collaborative task offloading scheme that can efficiently handle the request tasks with spatiotemporally uneven distribution in edge-computing-enabled transportation systems.
2) To reduce TL, we propose a node placement scheme based on an AGA. With the premise of minimizing TL, this scheme transmits the task requests from each sensor to an appropriate edge server.
3) To obtain reasonable task offloading decisions, we design a DRL scheme based on GNN that can learn the spatial features of task requests. On this basis, we present an efficient task forwarding strategy based on the greedy algorithm. It aims to facilitate efficient collaboration between edge servers, thereby overcoming system instability caused by sudden increases in task requests.
4) We perform comprehensive experiments with two real-world traffic data sets. The results show that TransEdge surpasses four baseline algorithms in terms of response latency and success rate.

This article is organized as follows. Section II provides a review of related work. Section III describes the system model and problem formulation. Section IV introduces the TransEdge. The proposed solution is evaluated in Section V. This article is concluded in Section VI.

II. Related Work

With the burgeoning of artificial intelligence (AI) and the emergence of edge computing, the transportation system has seen unprecedented development [1]. Thanks to the short distance between computing resources and users, edge computing is expected to alleviate the problem of limited capability of IoT sensors in transportation systems, while achieving the goal of low response latency (it is challenging to ensure low response latency just by using IoT sensors to process a large number of task requests [20]). Moreover, the introduction of AI technology further enables edge computing to learn knowledge from past experience and make intelligent task offloading decisions.

Huang et al. [21] considered using wireless power technology to charge terminal devices, and decomposed the task offloading decision and resource allocation into two subproblems. The above problems are tackled with online reinforcement learning. Bi et al. [22] described each time frame's decision as a multistage stochastic mixed integer nonlinear programming problem (MINLP). Subsequently, they combined the strengths of Lyapunov and DRL to design LyDROO to solve the above problem. The results show that LyDROO can effectively address the per-frame MINLP problem with remarkably low computational complexity and

is suitable for networks with fast-fading channels. Considering the impact of the multiple attributes of tasks, the inference accuracy of the model, and the inference error of the model, Fan et al. [23] proposed a machine learning task offloading scheme to minimize the total task processing latency with guaranteed task accuracy requirements. Given that both communication and machine learning computation cause data processing latencies and errors, Yang et al. [24] proposed a novel framework for machine learning task offloading based on edge computing, aiming to minimize total latency with factors, such as the complexity of the machine learning model and the error rate of inference, the quality of data, and the computational capacity of devices. However, as the number of edge servers rises, without a reasonable task offloading scheme among edge servers, it will cause intolerable latency and greatly reduce user satisfaction [25].

To achieve collaborative task offloading in edge computing, Lee and Lee [26] employed a traditional algorithm to orchestrate a collaboration between parked vehicles and traditional edge servers to establish a cooperative edge computing system, aiming to minimize the service latency of the entire vehicle group. Qi et al. [27] proposed a dual task offloading mechanism for the cooperation of a multiple unmanned aerial vehicles-assisted mobile-edge computing network. Guo et al. [28] proposed a multiple unmanned aerial vehicles collaborative communication and computing optimization scheme (MCCCO) and experimentally proved that MCCCO achieves better performance in reducing task processing latency and balancing UAV energy consumption load than traditional schemes. Yen et al. [29] devised a dual-layer system that involved the vehicle and the edge server, aiming to minimize cumulative execution cost by fine-tuning the choice of offloading destinations and the proportion of offloading. Pu et al. [30] suggested an auction algorithm for computational resource sharing, aiming to incentivize idle users to participate in task offloading, thereby facilitating cooperation among various users. Liu et al. [31] presented a task request schedule algorithm tailored to unmanned aerial vehicle swarms, which co-optimizes task offloading and multihop routing scheduling within an edge-cloud environment. These previous studies have successfully applied edge computing to various scenarios to deal with the challenges of resource constraints and latency reduction. However, for task requests that are unevenly distributed in time and space (e.g., the task request from transportation systems), it is hard for those studies to avoid the overloading problem of edge servers.

Recently, some researchers have tried to address the above issues. For example, Xu et al. [3] designed a task offloading approach for predicting traffic flow and minimizing the response latency in an Internet of Vehicles driven by edge computing. The authors first presented a graph weighted convolutional network (GWCN), aiming to mine the spatial feature among road segments to accurately predict traffic flow. The results from the prediction are used as the foundation for updating resource allocation across various regions. Subsequently, a deep deterministic policy gradient (DDPG)-driven task offloading approach is used to acquire the best task offloading solution for the edge server. Finally, comprehensive testing shows the superior performance of the proposed approach in reducing response latency. Liu et al. [19] considered two distinct scenarios: 1) off-peak periods and 2) peak periods, and the schemes executed in these two cases are different. During off-peak periods, tasks can be directly offloaded to MEC servers and the problem of minimizing average execution latency is modeled as the IP problem and addressed using the SAGA. During peak periods, task requests have the option of either being processed locally or offloaded to MEC servers, which is a more complex problem, so the author adopted the DQN to solve it. Despite the aforementioned schemes achieving good results, they still have high TL and a lack of reasonable task forwarding strategy, resulting in high response latency.

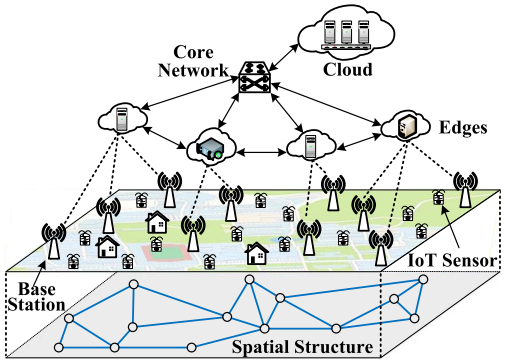

Fig. 1. System model of an edge-computing-enabled transportation system. It contains critical entities, such as the IoT sensor, the edge server, and the cloud server. Here, the IoT sensor is deployed beside the road to collect task requests from passing vehicles and transmit these task requests to the edge server. After receiving the task request, the edge server needs to judge whether to process it locally or offload it to other edge servers for processing. If all edge servers cannot handle the current task request, it will be offloaded to the cloud. Based on the connectivity of the road network, we can obtain the spatial structure among IoT sensors, which is the basis for learning the spatial features of each IoT sensor.

III. System Model and Problem Formulation

In this section, we first introduce the system model of the edge-computing-enabled transportation system. Then, the TL, computation latency (CL), and forwarding latency (FL) are modeled separately. Finally, we give the problem formulation and describe some key notations in this article.

A. System Model

Fig. 1 shows a scenario of an edge-computing-enabled transportation system, which includes one cloud server, L 5G base stations (BSs), M edge servers, and N IoT sensors. These servers are used to handle task requests (e.g., high-precision maps, music, video, etc.) from IoT sensors. The set of servers is defined as $C = \{c_0, c_1, \ldots, c_M\}$, where c_M is the cloud server, and the others are edge servers. The set of IoT sensors is denoted as $S = \{s_0, s_1, \ldots, s_{N-1}\}$, which is employed to monitor road traffic as well as collect and transmit task requests from passing vehicles. In particular, the BS and the server are connected through wired optical fiber, and the TL is negligible [32]. IoT sensors are connected to the BS via a

5G wireless link, and the latency is about 1 ms [33], [34]. For simplicity, we assume that the task offloading scheme operates in units of time slot t and consider that the network state is unchanged within this time slot, where $t \in [0, 1, \ldots, T]$, T is the life of the system cycle. The time slot t is usually a small value, which means that the vehicle is still within the coverage of the BS, and thus the mobility model of the transportation system can be ignored in this article [3], [19].

To maintain a high-quality user travel experience, our task offloading scheme should be able to dynamically adjust to adapt to accommodate randomly arriving task requests. We assume that the capacity set of all servers is set to $\text{Cap} = \{\text{cap}_0, \text{cap}_1, \ldots, \text{cap}_M\}$, indicating the maximum number of task requests each server can handle per second. cap_M represents the capacity of the cloud, which is far greater than the capacity of other edge servers. When servers still have the residual capacity, they can handle any arriving task requests through a reasonable forwarding rule. We define the total of task requests generated by passing vehicles as $R = \{r_0, r_1, \ldots, r_{N-1}\}$, where r_i represents the total task requests from IoT sensor s_i.

B. Latency Model

In general, the response latency of the edge-computing-enabled transportation system can be used to represent the user experience [35], i.e., the faster the system processes the task request, the smaller the response latency, and the better the user experience. The response latency in this article includes TL, CL, and FL. TL refers to the total latency of all task requests to be transmitted from IoT sensors to servers, as follows:

$$\text{TL} = \sum_{i=0}^{N-1} \sum_{j=0}^{M} r_i x_{ij} d_{ij} \tag{1}$$

where r_i represents the volume of task requests from IoT sensor s_i. x_{ij} is a binary decision variable, when $x_{ij} = 1$ indicates the IoT sensor s_i communicates with the server c_j and pushes all the task requests r_i collected by it to the server c_j for pending processing. d_{ij} represents the distance between IoT sensor s_i and server c_j.

CL represents the total latency generated by servers processing all task requests, as follows:

$$\text{CL} = \begin{cases} \sum_{i=0}^{N-1} \sum_{j=0}^{M} x_{ij} \frac{r_i}{\text{cap}_{y_{ij}}} & \text{if } r_i \le \text{cap}_{y_{ij}} \\ \sum_{i=0}^{N-1} \sum_{j=0}^{M} x_{ij} \frac{\text{cap}_{y_{ij}}}{\text{cap}_{y_{ij}}} & \text{if } r_i > \text{cap}_{y_{ij}} \end{cases} \tag{2}$$

where r_i is the same as r_i in (1). y_{ij} is a decision variable and satisfies $y_{ij} \in \{0, 1, \ldots, M\}$, which represents that the pending task request r_i on server c_j will be migrated to server $c_{y_{ij}}$ for processing. If $r_i \le \text{cap}_{y_{ij}}$, r_i is completely processed by server $c_{y_{ij}}$. Otherwise, only $\text{cap}_{y_{ij}}$ task requests are processed by the server on the server $c_{y_{ij}}$. The remaining task requests $r_i - \text{cap}_{y_{ij}}$ need to be forwarded to other servers for processing. Here, $\text{cap}_{y_{ij}}$ is the maximum processing capacity of server $c_{y_{ij}}$.

FL indicates the latency in transmitting all task requests from the current server to another server, as follows:

$$\text{FL} = \sum_{i=0}^{N-1} \sum_{j=0}^{M} x_{ij} r_i' d_{jy_{ij}} \tag{3}$$

where $r_i' = r_i - \text{cap}_{y_{ij}}$ indicates unprocessed task requests. $d_{jy_{ij}}$ represents the distance between server c_j and server $c_{y_{ij}}$.

Therefore, we can express the response latency as

$$\text{Latency} = \text{TL} + \text{CL} + \text{FL}. \tag{4}$$

C. Problem Formulation

This article aims to minimize the response latency of task requests in edge-computing-enabled transportation systems through rational cooperation among multiple edge servers. To this end, we can study the task offloading decisions problem with the objective of minimizing response latency in each slot t. Based on the above definitions, the task offloading decisions problem is formalized as

$$\textbf{P1}: \min_{\mathbf{x}^t, \mathbf{y}^t} \left(\text{TL}^t + \text{CL}^t + \text{FL}^t\right) \tag{5}$$

subject to:

$$\sum_{j=0}^{M} x_{ij}^t = 1 \quad \forall s_i \in S \tag{5a}$$

$$\sum_{i=0}^{N-1} r_i^t x_{ij}^t \le \text{cap}_{y_{ij}^t} \quad \forall c_{y_{ij}^t} \in C \tag{5b}$$

$$\sum_{i=0}^{N-1} r_i^t x_{ij}^t \le \text{cap}_j \quad \forall c_j \in C \tag{5c}$$

$$x_{ij}^t \in \{0, 1\}, y_{ij}^t \in \{0, 1, \ldots, M\} \quad \forall s_i \in S \ \forall c_j \in C \tag{5d}$$

where (5a) constraints on task offloading, and (5b) and (5c) are constraints on server capacity. Equation (5a) ensures that all task requests are processed. Equation (5b) ensures that the task requests migrated to each server do not exceed its maximum capacity. Equation (5c) ensures that the number of task requests does not exceed the maximum capacity of the server after tasks are offloaded to each server. Equation (5d) sets the value range of the decision variable. Moreover, we define $\mathbf{x}^t = \{x_{0,0}^t, \ldots, x_{N-1,M}^t\}$ and $\mathbf{y}^t = \{y_{0,0}^t, \ldots, y_{N-1,M}^t\}$. A summary of the key notations is provided in Table I.

IV. TransEdge Overview

P1 is a constrained mixed integer linear programming problem (MILP), and it is also an NP-hard problem [36], which cannot be solved in polynomial time. To this end, we propose a novel framework named TransEdge, to solve the above MILP problem while maintaining minimal response latency in edge-computing-enabled transportation systems. Specifically, we first propose a node placement scheme based on an AGA to achieve reasonable placement of IoT sensors, aiming to minimize TL. Subsequently, a DRL framework based on the GNN is designed to make the decision of task offloading. Finally, we introduce a task forwarding strategy based on the greedy algorithm to facilitate collaboration

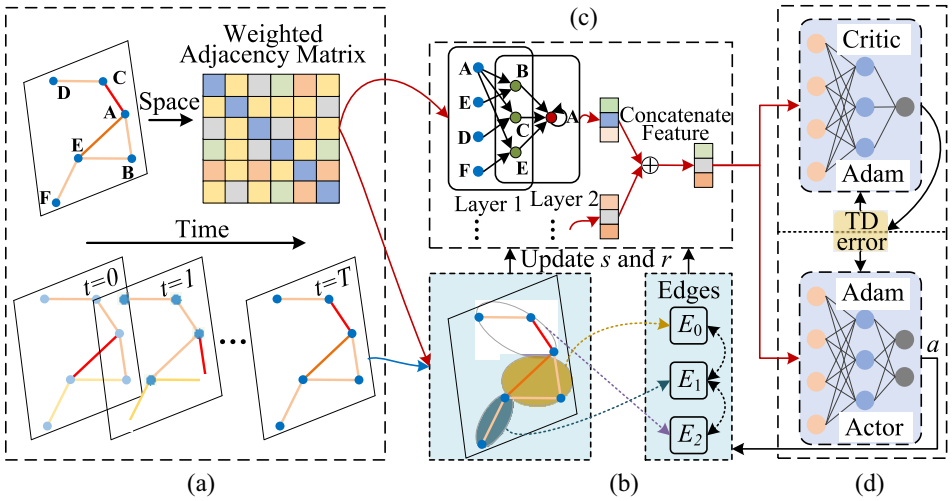

Fig. 2. TransEdge overview. In (a), spatiotemporal data consists of spatial data between sensors in the transportation system (i.e., weighted adjacency matrix) and time-varying task request data, which is the input of TransEdge. In (b), we obtain the placement scheme of sensor nodes based on the AGA. This scheme can preassign each sensor node to a specific server. Furthermore, the task forwarding strategy based on the greedy algorithm facilitates collaborative task offloading among edge servers. In (c), a GNN is used to capture the spatial features in the transportation system, and the richer information composed of this feature and other environmental states is used as the input of reinforcement learning. In (d), the Actor–Critic interacts with the environment and makes reasonable task offloading decisions. Also, it adjusts its policy based on TD-error, aiming to make better decisions in the future.

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
N	Number of IoT sensors
M	Number of edge servers
r_i	The total task requests from IoT sensor s_i
x_{ij}	A binary decision variable indicating whether the task request r_i from IoT sensor s_i is offloaded to server c_j
d_{ij}	The distance between IoT sensor s_i and server c_j
y_{jk}	The percentage of the task request r_i on c_j that is offloaded to the server c_k
cap_k	The maximum processing capacity of the kth server
TL	The total latency of all task requests to be transmitted from IoT sensors to servers
CL	The total latency generated by servers processing all task requests
FL	The latency in transmitting all task requests from the current server to another server
sf_j^l	The spatial feature of IoT sensor s_j at the lth layer
N_i	The set of neighbors of IoT sensor s_i
c_{ij}	The value of the square root of the degree product of s_i and s_j
$RCap_i^t$	The remaining capacity set of the server after processing the task request of s_i at time slot t
A_{pro}	The logarithmic probability of the Actor executing a specific action

among edge servers. The framework of TransEdge is shown in Fig. 2.

A. Node Placement Scheme Based on Adaptive Genetic Algorithm

Given **P1** is difficult to solve directly, we consider solving its subproblems separately. In this section, we need to minimize TL^t in **P1**. According to (1), the TL optimization problem **P2** is defined as

$$\mathbf{P2}: \min_{\mathbf{x}^t} \left(\sum_{i=0}^{N-1} \sum_{j=0}^{M} r_i^t x_{ij}^t d_{ij} \right) \tag{6}$$

subject to: (5a), (5c), (5d).

Theorem 1: The TL optimization problem **P2** is NP-hard.

Proof: We prove the NP-hardness of **P2** by reducing from the multiple knapsack problem (MKP), which is known to be NP-hard [37]. Given N items and M knapsacks, each item i has a weight w_i and a value v_i. Each knapsack j has a maximum capacity W_j. The MKP selects suitable items put into knapsacks, ensuring that the total capacity of each knapsack does not exceed its maximum capacity while maximizing the total value of the items in all knapsacks. In **P2**, the IoT sensor s_i corresponds to the item i in the MKP, the task request r_i corresponds to the item weight w_i in the MKP, the distance d_{ij} represents the value of the item in the MKP and cap_j corresponds to the maximum capacity W_j in each backpack in the MKP. It can be seen that the solution to **P2** is also the optimal solution to the MKP. Therefore, **P2** is an NP-hard. ■

To solve problem **P2**, we propose a node placement scheme based on an AGA. The adaptability of AGA is reflected in 1) adaptively adjusting the population to obtain better evolved individuals; 2) adaptively update the crossover probability, aiming to expand the exploration space without reducing search efficiency; 3) adaptively update the mutation probability, reducing search randomness and accelerating the convergence; and 4) adaptively exiting the search, to avoid overfitting and save computational resources. Fig. 3 shows the process of AGA, which is divided into five steps, as follows.

1) *Step 1 (Generate Population):* A 2-D array of $Popsize \times N$ is generated as the population P of AGA, where $Popsize$ is the size of the population, and N is the scale of a single node placement solution, which is equal to the number of IoT sensors. The values in the 2-D array are integers between 0 and M, which indicates gene encoding and also indicates which server the IoT sensor should be assigned to.

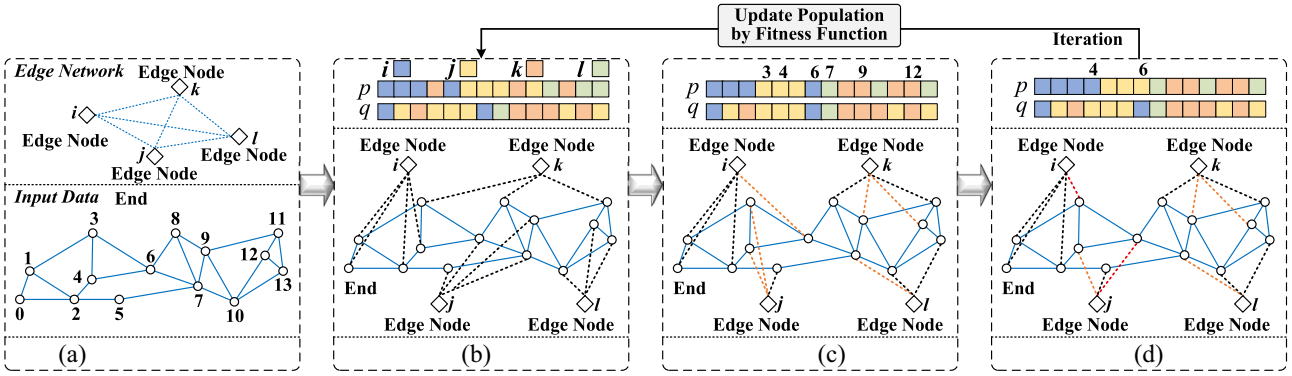

Fig. 3. Instance of AGA for node placement scheme. In (a), an edge network and input data are given. In (b), AGA selects individuals as parents from the population by natural selection, e.g., p and q, which are solutions to node placement, representing which IoT sensor is associated with which edge server. Given the constraints of space, only the results for p are presented in this figure. In (c), the parents cross over with a certain probability to achieve the combination of genes, thus obtaining new individuals. In (d), AGA realizes the mutation of its own genes based on the new individuals generated in (c), enriching the diversity of the population and ensuring that AGA does not fall into the optimal. AGA needs to repeat iterations from (b) to (d) until convergence to obtain better solutions.

2) *Step 2 (Natural Selection):* In each iteration, AGA selects superior individuals from the population P to evolve through a roulette wheel selection method [38]. This means that individuals with higher fitness have a greater probability of being selected as parents. Fig. 3(b) shows that AGA selects two individuals p and q for evolution. The blue block, yellow block, orange block, and green block represent server i, server j, server k, and server l, respectively. Due to space limitations, we only show the results of p.
3) *Step 3 (Crossover):* Based on the individual selected in step 2, AGA completes the crossover between individuals with the crossover probability $Crossover_{prob}$, achieving the combination of different genes to obtain new individuals. As shown in Fig. 3(c), p and q perform the crossover, and the 3rd, 4th, 6th, 7th, 9th, and 12th genes of p are updated. The yellow line represents the node placement results after updating individual p.
4) *Step 4 (Mutation):* Based on the individual updated in step 3, AGA mutates the genes of individuals themselves with a mutation probability of $Mutation_{prob}$, aiming to obtain new individuals and store these new individuals into P'. As shown in Fig. 3(d), the 4th and 6th genes of the individual p have mutated. The red line shows the node placement solution after updating individual p.
5) *Step 5 (Update and Early Stopping): Update Population:* AGA merges the populations P and P' and selects the top *Popsize* individuals as the new population P based on (6).

Update Crossover Probability: The crossover probability aims to enhance the diversity of the new generation population. Dynamically adjusting the crossover probability for AGA allows it to broadly explore the solution space without reducing its search efficiency. The equation for dynamically adjusting the crossover probability as

$$Crossover_{prob} = \text{mean}\left(\sum_{j=0}^{M} r_i^t x_{ij}^t d_{ij} / \sum_{i=0}^{N-1}\sum_{j=0}^{M} r_i^t x_{ij}^t d_{ij}\right). \tag{7}$$

Update Mutation Probability: The mutation probability is intended to ensure that the genetic algorithm does not stagnate at the local optimal solution. However, an unreasonable mutation probability may cause the search process of the algorithm to become random, making it difficult to converge to the optimal solution. To this end, we implement the dynamic adjustment of the mutation probability, as follows:

$$\text{Mutation}_{\text{prob}} = \max\left(\sum_{j=0}^{M} r_i^t x_{ij}^t d_{ij} / \sum_{i=0}^{N-1}\sum_{j=0}^{M} r_i^t x_{ij}^t d_{ij}\right). \tag{8}$$

Early Stopping: Early stopping can avoid overfitting while saving computational resources. For example, if the difference between the optimal fitness value and the average fitness value is smaller than a threshold (e.g., 0.001) many times (e.g., ten times), it can be considered that the algorithm has converged. Exiting computation at this time can prevent the waste of computational resources.

By repeating steps 2–5 many times, we can obtain a solution that satisfies **P2**.

B. Collaborative Task Offloading Scheme Based on Graph Reinforcement Learning

In Section IV-A, we have obtained the solution of **P2**. Next, we define the other subproblems of **P1** as

$$\mathbf{P3}: \min_{\mathbf{y}^t} \left(\text{CL}^t + \text{FL}^t\right) \tag{9}$$
$$\text{subject to: (5b)}-\text{(5d)}.$$

P3 denotes how to assign appropriate task requests to each server to minimize the sum of CL and FL. It can be seen from (2) that the size between the number of task requests and the capacity of the destination server determines whether the task forwarding operation is required. Therefore, there may be two different expressions of **P3**.

1) When the capacity of the destination server is sufficient to handle the arriving task requests, at which time there

is no requirement to forward the task request to other servers, i.e., $FL^t = 0$. Therefore, **P3** can be rewritten as

$$\textbf{P3-1}: \min_{\mathbf{y}^t} \left(\sum_{i=0}^{N-1} \sum_{j=0}^{M} x_{ij}^t \frac{r_i^t}{\text{cap}_{y_{ij}^t}} \right) \quad \text{subject to: (5b)}-\text{(5d)}. \tag{10}$$

2) When the capacity of the destination server cannot handle all arriving task requests, it is necessary to migrate the remaining task requests to other servers for processing, and the FL is as shown in (3). Therefore, **P3** can be rewritten as

$$\textbf{P3-2}: \min_{\mathbf{y}^t} \left(\sum_{i=0}^{N-1} \sum_{j=0}^{M} x_{ij}^t \frac{\text{cap}_{y_{ij}^t}}{\text{cap}_{y_{ij}^t}} + \sum_{i=0}^{N-1} \sum_{j=0}^{M} x_{ij}^t r_i'^t d_{jy_{ij}^t} \right) \quad \text{subject to: (5b)}-\text{(5d)}. \tag{11}$$

The interdependence of decisions in **P3-1** and **P3-2** makes finding an optimal solution extremely difficult. To solve the aforementioned problem, we design a task offloading method using the GNN and Actor–Critic, and a task forwarding strategy based on the greedy algorithm. Fig. 4 gives an example of these two schemes to solve **P3-1** and **P3-2**. Next, we will introduce these schemes in detail.

1) Task Offloading Scheme Based on Graph Neural Network and Actor–Critic: Different from other fields, the IoT sensors in edge-computing-enabled transportation systems affect each other, leading to task requests with spatial features. Traditional DRL cannot learn this feature [39], [40]. Fortunately, the advent of GNNs makes it possible to learn spatial features contained in task requests [41], [42]. To this end, we hope to improve DRL with the GNN, given it the ability to capture the spatial features, thereby improving the accuracy of its task offloading decision.

Fig. 2(c) shows the two-layer operation of the GNN, the process can be expressed as

$$sf_i^{l+1} = \sigma \left(\sum_{j \in N_i} \frac{1}{c_{ij}} sf_j^l w^l \right) \tag{12}$$

where N_i represents the set of neighbors of IoT sensor s_i. c_{ij} denotes the value of the square root of the degree product of s_i and s_j. sf_j^l represents the spatial feature of s_j at the lth layer. w is a trainable weight parameter. σ is the activation function. Equation (12) signifies that each node's spatial features are updated through a weighted average of the information from its own and its neighbors.

On the basis of spatial features, we introduce the Actor–Critic reinforcement learning framework to predict the decision of task offloading, as shown in Fig. 2(d). Next, we will introduce the framework in detail.

1) *Actor–Critic Framework:* The Actor–Critic is highly efficient in solving the problem of task offloading between edge servers [43], [44]. In this article, the Actor is a fully connected layer with an input dimension of 128 and an output dimension equal to the number of servers, which is utilized to determine the task offloading decision. The Critic is also a fully connected layer with an input dimension of 128 and an output dimension is 1, which is employed to assess the current task offloading decision. Besides, there is a common fully connected layer, whose input dimension is the sum of the number of IoT sensors, the number of servers, and the spatial feature of each IoT sensor plus 1, and the dimension of the output layer is 128. The common fully connected layer acts as a shared layer for the Actor and Critic. We hope that the Actor can make better decisions and enable the Critic to give better scores to corresponding decision making through training.

2) *Action Space and State Space:* According to the AGA scheme in Section IV-A, we know that the task request on the IoT sensor will first be transmitted to the corresponding server for processing (either processed on the current server or offloaded to other servers). At each time slot t, the Actor–Critic processes the task request on each IoT sensor one by one. The Actor determines where to process the task requests of the IoT sensors based on the

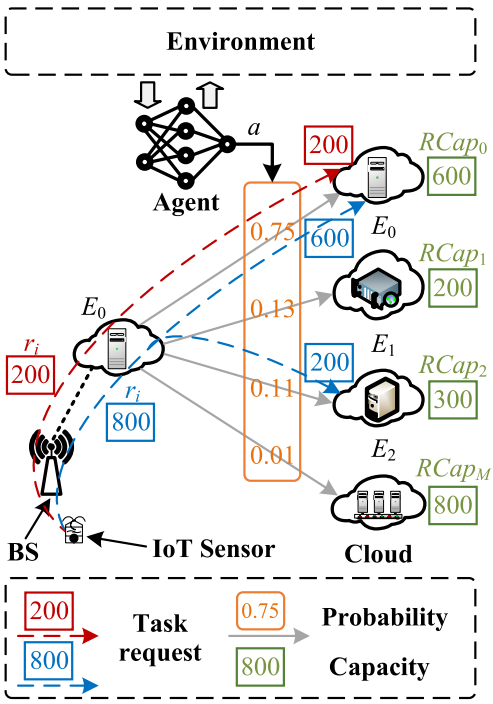

Fig. 4. Example of solving **P3-1** and **P3-2**. In this example, the task requests (r_i) generated by the IoT sensor are offloaded to the server E_0 via the BS to wait for processing. Next, the agent (composed of a neural network) interacts with the environment (including information, such as remaining task requests, remaining capacity of the server, and spatial feature between IoT sensors) to generate an action policy (a), i.e., the probability of offloading task request to each server (i.e., 0.75, 0.13, 0.11, and 0.01) and the server with the largest probability value is selected as the destination server (i.e., E_0). Finally, task requests are allocated reasonably based on the size relationship between the task request volume and the remaining capacity of the server (*RCap*). Specifically, when the number of task requests is 200 (red dotted), the remaining capacity of server E_0 is sufficient, and all task requests can be processed using only server E_0. When the number of task requests is 800 (blue dotted), server E_0 can handle up to 600 task requests, and the remaining 200 task requests need to be migrated to a reasonable server (i.e., E_2) for processing through our task forwarding strategy to improve user experience.

current state of the environment. Here, the action space of the task offloading decision is $[0, 1] \times (M+1)$, which represents the probability that the current task request is offloaded to $M+1$ servers for processing. (In this article, we select the server corresponding to the maximum probability value as the destination server.) The state can be expressed as $State^t = \{RR^t, RCap^t, SF^t\}$. RR^t is the remaining task request. $RCap^t$ represents the remaining capacity set of all servers. SF^t is the spatial features of all IoT sensors learned by the GNN.

3) *Reward Function:* In reinforcement learning, an agent obtains a certain reward when it performs an action. The objective of the Actor–Critic model is to optimize the cumulative reward through training to obtain an appropriate computation offloading scheme. At time slot t, the task request of IoT sensor s_i is offloaded to servers, whose reward can be expressed as

$$\text{reward}^t = -\frac{\text{CL}_i^t}{\text{Cap} - \textit{R}\text{cap}_i^t} - \sum_{q=0}^{\text{FT}_q^t} \gamma \tag{13}$$

where γ represents the penalty for the forwarding operation. $\textit{R}\text{cap}_i^t$ represents the remaining capacity set of the server after processing the task request of s_i at time slot t. According to (2), CL_i^t indicates the CL for processing task request r' at time slot t. FT_q^t indicates the number of forwarding times when processing task request r_i^t at time slot t. Reinforcement learning usually aims to maximize reward, while our goal is to minimize response latency. Therefore, we need to take the negative value of the reward.

4) *Loss Function:* The temporal difference error (TD-error) represents the discrepancy between the estimated value of the current state and the actual reward obtained [45]. In the Actor–Critic algorithm, the result generated by the Critic is used to calculate the TD-error. The TD-error is often used to update the value estimation of the Critic itself and is also used as a signal to guide the adjustment of the Actor's policy. After continuous iteration, the policy generated by the Actor will tend to choose actions that can reduce the TD-error, which is better than expected. The TD-Error plays a bridging role in the learning process of the Actor and Critic, enabling them to learn collaboratively and continuously improve the estimation of policy and value functions to achieve better performance. For this purpose, we devise a loss function based on TD-error to update the neural network, which can be expressed as

$$\text{Loss} = -A_{\text{pro}} \times \delta + \delta^2 \tag{14}$$

where A_{pro} is the logarithmic probability of the Actor executing a specific action. TD-error represents $\delta = \textit{reward} + 0.99 \times \textit{next\_value} - \textit{value}$. *reward* is calculated based on (13). *value* and *next_value* are calculated by the Critic based on the current state and the next state.

2) Task Forwarding Strategy Based on Greedy Algorithm: In general, task requests are generated randomly. When there are fewer task requests, only a single server can handle all requests. In this case, the solution given by the above task offloading algorithm based on GNN and DRL is applicable. However, with the increase of task requests, especially the sudden increase of task requests (common in transportation systems), it is easy to cause an increase in response latency due to the limited capacity of a single server. Therefore, it is necessary to design a reasonable strategy to migrate the remaining task requests to the appropriate servers for processing. To this end, we propose a task forwarding strategy based on the greedy algorithm, as described in Algorithm 1. The core idea of Algorithm 1 is that when the remaining capacity of the current server cannot handle all incoming tasks, some tasks need to be forwarded to a server with excellent processing performance and still has the remaining capacity, thus ensuring lower response latency.

Algorithm 1 Task Forwarding Strategy Based on Greedy Algorithm

Input:
r' // the task request that needs to be processed
Cap // the capacity set of servers
M // the number of edge servers
Output:
CF // the sum of CL and FL based on (9)

1: $CF \leftarrow 0$, $RCap \leftarrow Cap$ // $RCap$ represents the remain capacity set of server;
2: **repeat**
3: **if** $\sum_{i=0}^{M-1} \neq 0$ **then** // Edge servers have capacity;
4: $Id \leftarrow Find(RCap[:M])$; // Find the edge server with the best capacity based on greedy algorithm;
5: **if** $RCap[Id] \geq r'$ **then**
6: $CF \leftarrow CF + r'/Cap[Id]$;
7: $RCap[Id] \leftarrow RCap[Id] - r'$;
8: $r' \leftarrow 0$;
9: **else**
10: $CF \leftarrow CF + RCap[Id]/Cap[Id]$;
11: $r' \leftarrow r' - RCap[Id]$;
12: $RCap[Id] \leftarrow 0$;
13: **end if**
14: **else**// Only the cloud server has the capacity;
15: $CF \leftarrow CF + r'/Cap[M]$;
16: $RCap[M] \leftarrow RCap[M] - r'$;
17: $r' \leftarrow 0$;
18: **end if**
19: **until** $r' = 0$
20: **return** *CF*

Algorithm 1 accepts the task request r', the capacity set of servers *Cap*, and the number of servers M as input, aiming to optimize the solution of **P3-2**, i.e., to minimize the sum of CL and FL. The r' comes from the remaining tasks after offloading using reinforcement learning. We define the sum of CL and FL as *CF* and the remaining capacity set of servers as *RCap*, and initialize them. As long as the r' is not 0, Algorithm 1 needs to continue forwarding the task until it is completely processed (line 2).

After the task request r' arrives, Algorithm 1 first judges whether edge servers still have the remaining capacity. If not, they directly offload the task request r' to the cloud (line 14) update the task of r', the remaining capacity set of the server, and return the result. Otherwise (line 3), Algorithm 1 will use the greedy algorithm to find the best-performing edge server among those with remaining capacity, and use it to process the current task request r' to ensure the minimum CF (line 4). However, not the remaining capacity of each edge server can process the task request r' at once. Algorithm 1 needs to judge whether the selected edge server can process the task request r' at one time. If it can (line 5), it updates the task request r' and the remaining capacity set of the server, and returns the result. Otherwise (line 9), after updating task request r' and the remaining capacity set of the server, Algorithm 1 still needs to continue using the greedy algorithm to find a suitable server to process r'.

The main operations of Algorithm 1 include loop computation (line 2) and Find function (line 4). The Find function is implemented based on the greedy algorithm, and it needs to traverse M servers, so its time complexity is $O(M)$. The main loop of Algorithm 1 is the repeat loop (lines 2–19). In the worst case, the value of task request r' (we assume the value is Z) is decremented by a constant in each iteration, so this loop will repeat Z times and the time complexity of this loop is $O(Z)$. Therefore, the time complexity of Algorithm 1 is $O(M) \times O(Z)$, i.e., $O(M \times Z)$. In fact, the number of M is usually less than 100, or even less than 10. At each time slot t, the value of the task request r' (i.e., Z) is usually also around 10. In conclusion, although the time complexity of Algorithm 1 is $O(M \times Z)$, it is still within the acceptable range.

V. Experiments

In this section, we outline the experimental setups and discuss the results. The results obtained with TransEdge are compared with four algorithms for task offloading in edge-computing-enabled transportation systems. The source code of TransEdge is available in [46].

A. Experimental Setups

Data set: We validate the proposed model on two highway traffic data sets PEMS4 and PEMS8, which come from the real world and are widely used in the traffic field [47]. PEMS4 represents traffic data from the San Francisco Bay Area, which contains 307 IoT sensors and spans from January to February 2018. PEMS8 contains traffic data from San Bernardino for July–August 2016, including 170 IoT sensors. In this article, we select data of 1000 consecutive time slots (approximately 3.5 days, where each time slot represents the number of task requests generated per 5 min) from the data set for experimental verification, as shown in Fig. 5. Fig. 6 shows the arrival of task requests from two different sensors in PEMS8.

Parameter Settings: All the results are evaluated on Intel Xeon Silver 4112 2.2-GHz CPU and 13 GB of memory. The population size *Popsize* in AGA is 100. The GNN utilized in this article is composed of two layers. The input dimension of the first layer is determined by the data set's dimensions (2816 for PEMS8, and 2624 for PEMS4), and the output dimension is 128. The input dimension of the second layer is 128, and the output dimension is 6. The number of edge servers M is 5. N is determined by the data set, N is 170 in PEMS8, and N is 307 in PEMS4. σ in (12) is the activation function *ReLU* [48] and γ in (13) is 0.03. The value of *Cap* is determined by the data set, whose unit is task request per second (i.e., the number of task requests processed by the edge server per second) [49]. When evaluating TransEdge on PEMS8, $Cap = \{400, 50, 100, 50, 200, 1000\}$, when evaluating TransEdge on PEMS4, $Cap = \{800, 50, 100, 50, 300, 1000\}$.

Baseline Algorithms: This article demonstrates the advantages of TransEdge by comparing it with four baseline algorithms, namely, GWCN [3], GADQN [19], Near, and Random. GWCN is the first to achieve collaborative task offloading in transportation systems where task requests are unevenly distributed in space and time. It utilizes the GNN to estimate the number of task requests in the future and preallocate corresponding resources, while implementing task request scheduling between edge servers based on DDPG. Similar to GWCN, GADQN also considers task requests in transportation scenarios. Differently, GADQN considers two

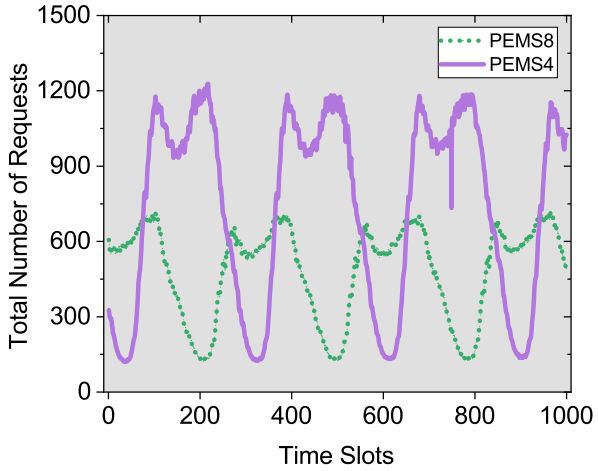

Fig. 5. Example of the total task requests for each time slot in two data sets.

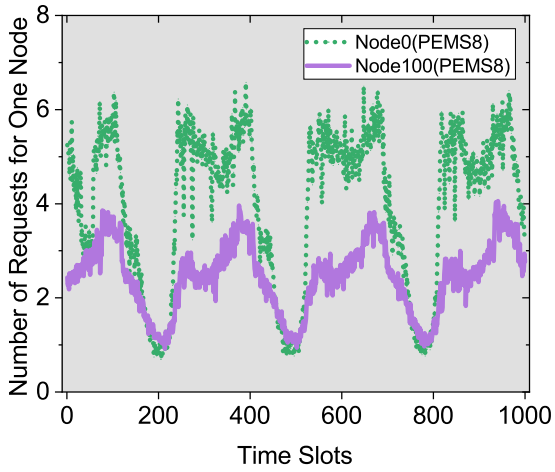

Fig. 6. Example of the task requests for each time slot on two sensors in PEMS8.

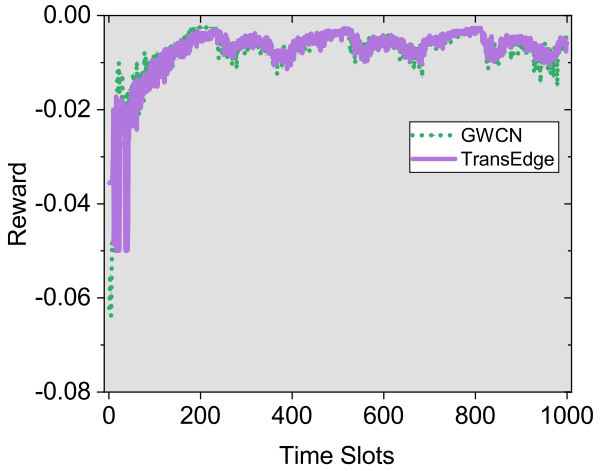

Fig. 7. Comparison of the reward.

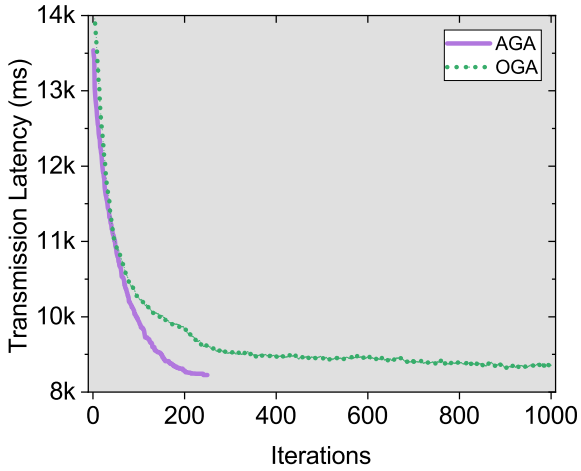

Fig. 8. Performance comparison of AGA and OGA.

situations: 1) during off-peak periods, it considers offloading tasks directly to edge servers and models this problem as an IP, while using SAGA to solve this problem and 2) during peak periods, it uses DQN to determine whether to process task requests locally or offload them to edge servers. Near implies that task requests from sensors are only transmitted to the nearest edge servers for processing. Random denotes that the task request of the sensor is randomly transmitted to any edge server for processing.

Evaluation Metrics: We evaluate the TransEdge with two metrics, including the response latency and success rate. The response latency is a key indicator in evaluating the edge-computing-enabled transportation system [50]. Considering that the generated tasks can be processed using either edge servers or central cloud servers, where the latter usually incur high response latency to meet the real-time requirements of terminal users and can be regarded as failures. To this end, we define the success rate as

$$\text{SuccessRate}^t = 1 - \frac{r^t_{\text{cloud}}}{\sum_{i=0}^{N-1} r^t_i} \tag{15}$$

where r^t_{cloud} denotes the total number of task requests processed by the central cloud server in the tth time slot.

B. Results and Analysis

1) Convergence of TransEdge: As shown in Fig. 7, we evaluate the convergence of TransEdge. The two curves in the figure represent the reward functions of the state-of-the-art algorithm GWCN and our proposed scheme, respectively. It can be seen that TransEdge has a similar convergence performance to GWCN. Also, their reward gradually increases over time, and gradually flattens after 200 time slots, but there are still minor fluctuations. This is because the transportation scenario is dynamically changing and the volume of task requests arriving also varies. Since the reward value is related to the task request volume, the reward value will fluctuate within a small range.

2) Rationality of AGA in TransEdge: To explore the rationality of AGA, we introduce the original genetic algorithm (OGA) in TransEdge to compare with TransEdge with AGA. Fig. 8 shows that the AGA scheme is superior to OGA in terms of convergence speed and TL. This is because AGA adopts adaptively adjusted crossover probability and mutation probability, enabling it to generate solutions that adapt to the current state more optimally and stably. Furthermore, the mechanism of updating the population in AGA makes it possible to dynamically retain the current optimal solution. Last but not least, the introduction of an early stopping mechanism allows it to exit the optimization phase faster while avoiding the waste of computational resources.

3) Performance Comparison Under Long-Term Continuous Constraints: We compare the results of TransEdge with four baseline algorithms (i.e., Random, Near, GADQN, and GWCN) on PEMS8 and PEMS4, as shown in Figs. 9–11. Overall, TransEdge outperforms all schemes in performance after convergence (i.e., after the 200th time slot in Fig. 7) on the two real data sets, demonstrating the advantages of our proposed scheme. Specifically, compared to the state-of-the-art algorithm (i.e., GWCN), TransEdge improves response latency performance by at least 9.3% and 3.7% on the PEMS8 and PEMS4 data sets. Moreover, we can also make the following observations.

1) The scheme based on reinforcement learning (i.e., GADQN, GWCN, and TransEdge) is better than the traditional scheme (i.e., Random and Near). This is because the reinforcement learning-based schemes, after a long period of interaction with the environment, have learned how to deal with the sudden increase in task requests.
2) In overall performance, Random always has the highest latency because it cannot find suitable edge servers for task request transmission and forwarding, i.e., it increases the probability of forwarding task requests to the central cloud server and reduces the success rate. Although Near has the lowest FL, due to the sudden increase of task requests, it only forwards task requests to the server closest to itself, resulting in a high CL.
3) In partial performance, the TL of TransEdge is better than other schemes, which is attributed to AGA being able to obtain a better node placement solution in a

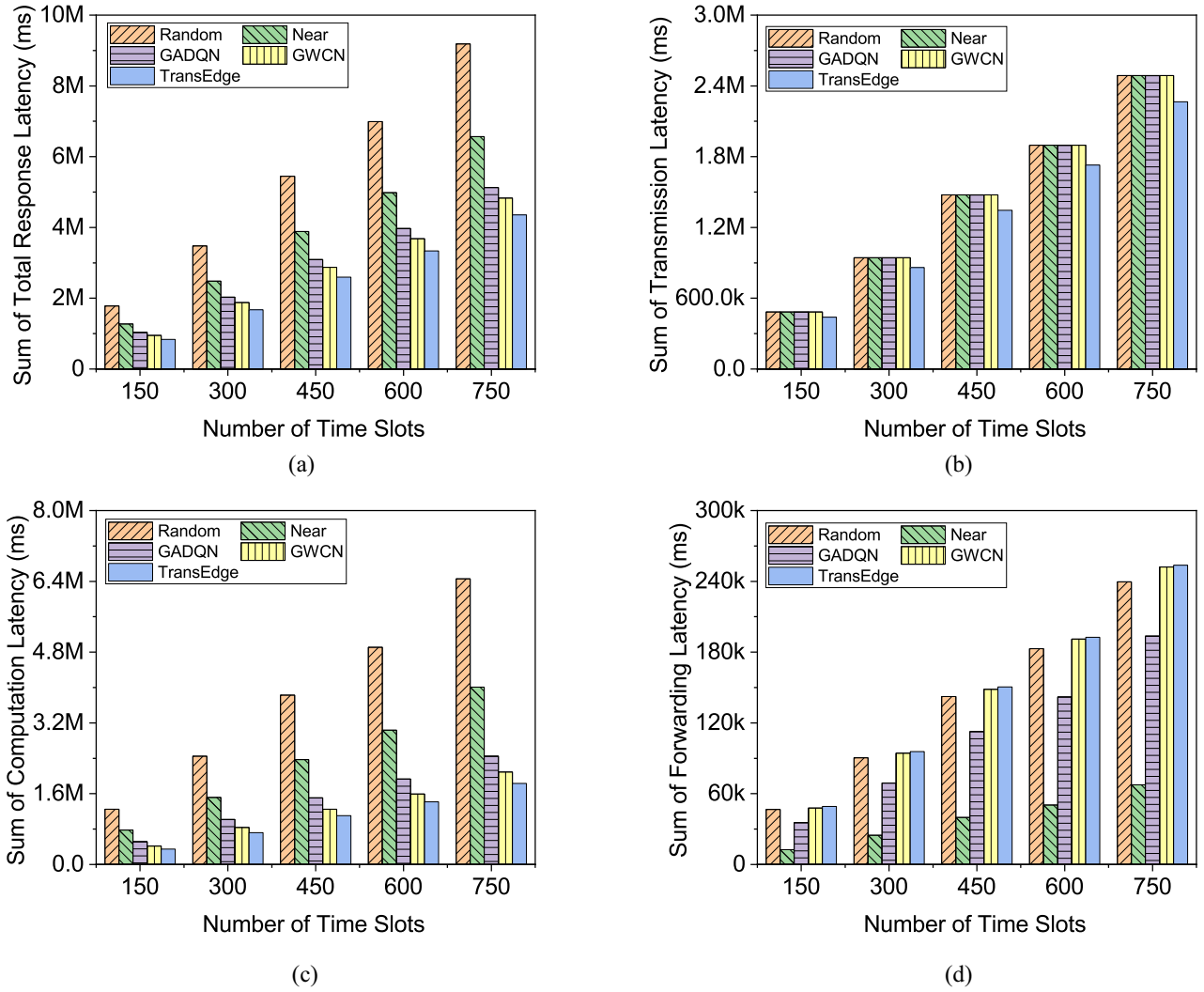

Fig. 9. Latency comparison of different algorithms under long-term continuous constraints on PEMS8.(a) Comparison of the total response latency. (b) Comparison of the TL. (c) Comparison of the CL. (d) Comparison of the FL.

short time. Compared with other solutions, even though TransEdge needs more FL to achieve task scheduling, it can always find the servers with better capacity for processing task requests, hence its CL is lower. Last but not least, TransEdge has the highest success rate (this value is close to 99%), which means that it offloads fewer task requests to the central cloud server for processing, effectively reducing response latency.

4) Performance Comparison Under Different Capacity: As shown in Figs. 12 and 13, we evaluate the performance of all schemes by varying the total capacity of all edge servers. The results indicate that with the increase in the total capacity of the edge server, the average response latency of all schemes will decrease and the success rate of them will increase. The reasons are: 1) the better the edge server capacity, the shorter the CL for processing the same task request and 2) as the capacity of edge servers increases, the probability of forwarding task requests to the central cloud server gradually decreases, thereby improving the success rate of task request processing. Particularly, the reinforcement learning-based solutions (i.e., TransEdge, GWCN, and GADQN) outperform Random and Near in handling sudden task requests. The reason is that reinforcement learning enables the agent to learn from experience gained through interacting with the environment and improve based on TD-error, thus endowing the agent with the ability to cope with sudden task requests. Specifically, GADQN considers the off-peak period and the peak period separately and its performance in the off-peak period is similar to Near, and the performance in the peak period is similar to GWCN. Different from them, TransEdge uses AGA for efficient IoT sensor placement during the transmission process and employs Actor–Critic for task offloading after capturing the spatial features among IoT sensors through GNNs. Last but not least, when the server cannot handle the current task request, a more flexible and efficient greedy strategy is employed for task forwarding. The above strategies make TransEdge achieve the best performance. Particularly, compared with GWCN, our TransEdge improves the average response latency by 3.5%–9.5% (on PEMS8) and 4.7%–9.4% (on PEMS4).

5) Performance Comparison Under Different Resource Placement: Fig. 14 shows that as the total capacity of all edge

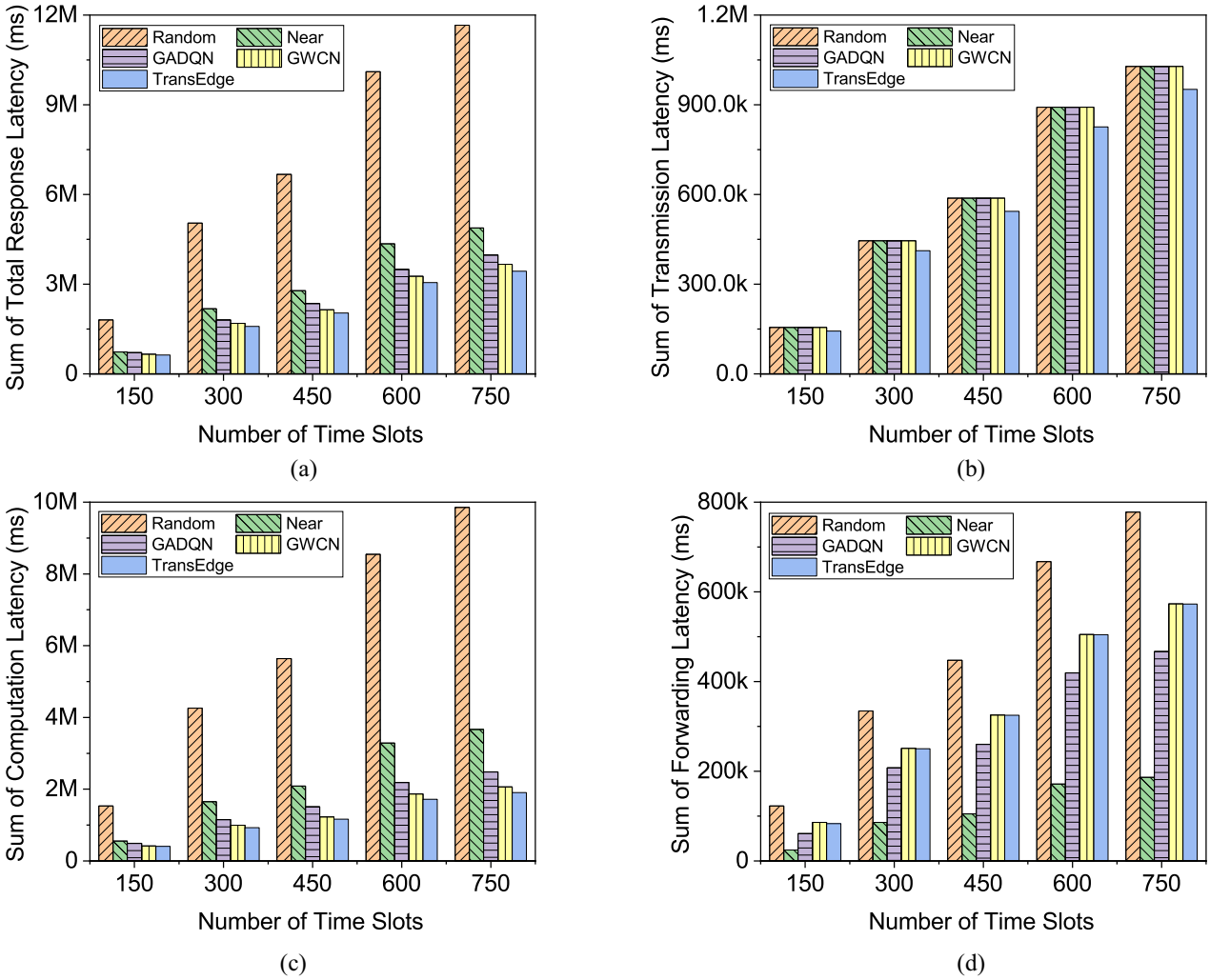

Fig. 10. Latency comparison of different algorithms under long-term continuous constraints on PEMS4. (a) Comparison of the total response latency. (b) Comparison of the TL. (c) Comparison of the CL. (d) Comparison of the FL.

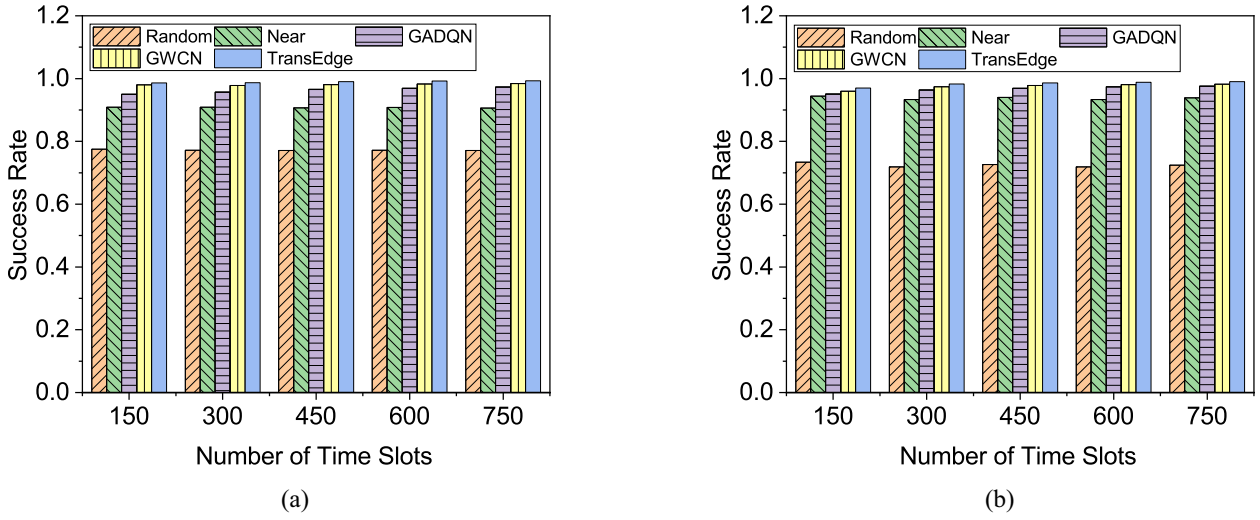

Fig. 11. Success rate comparison of different algorithms under long-term continuous constraints. (a) Success rate comparison on PEMS8. (b) Success rate comparison on PEMS4.

servers increases, the response latency of some methods (e.g., Random and Near) does not necessarily decrease. This motivates us to explore the impact of different resource placements on the performance of all schemes, as shown in Fig. 14. It can be seen that despite changing resource placement, TransEdge still has the lowest response latency compared

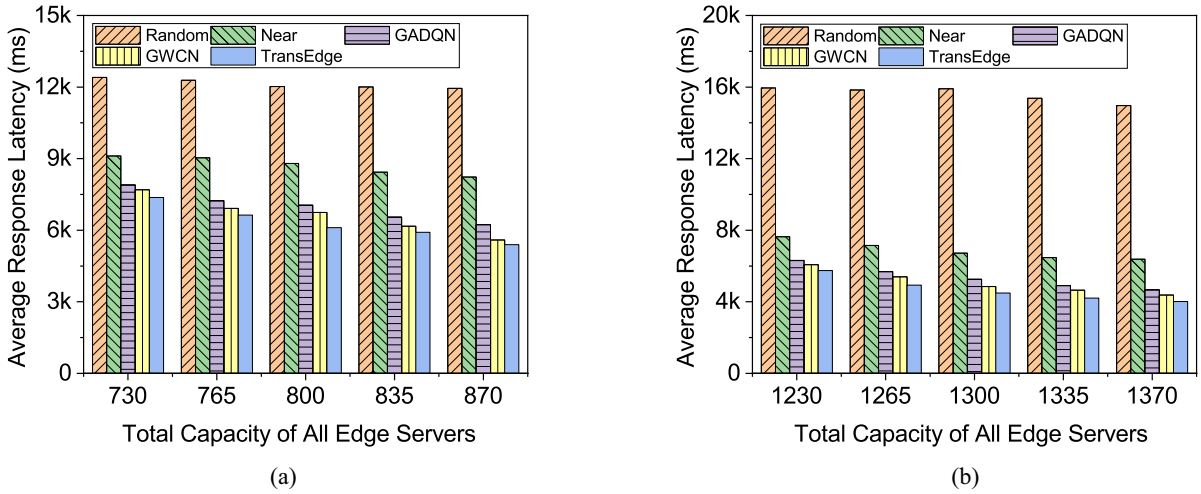

Fig. 12. Latency comparison of different algorithms under different capacity. (a) Latency comparison on PEMS8. (b) Latency comparison on PEMS4.

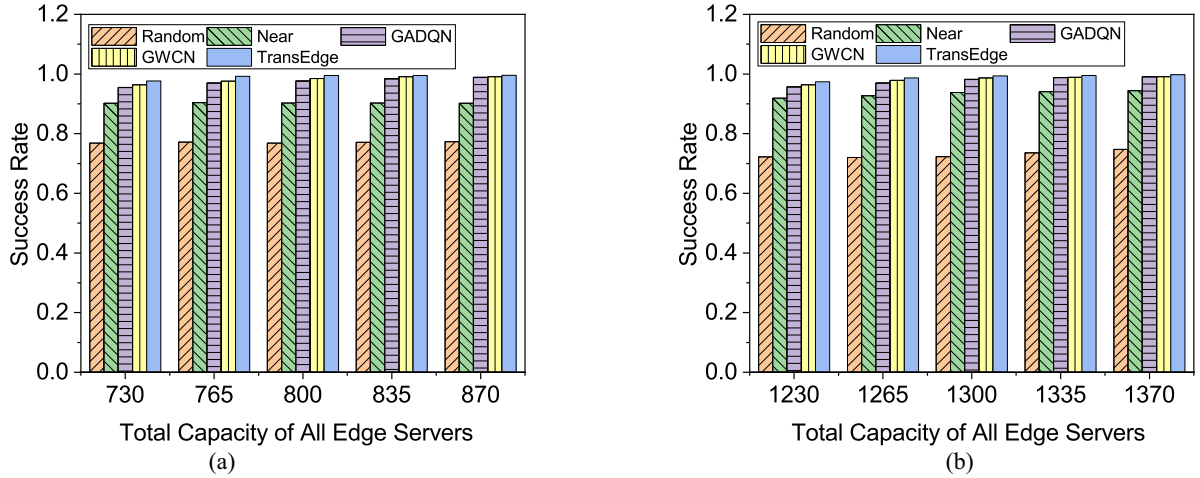

Fig. 13. Success rate comparison of different algorithms under different capacity. (a) Success rate comparison on PEMS8. (b) Success rate comparison on PEMS4.

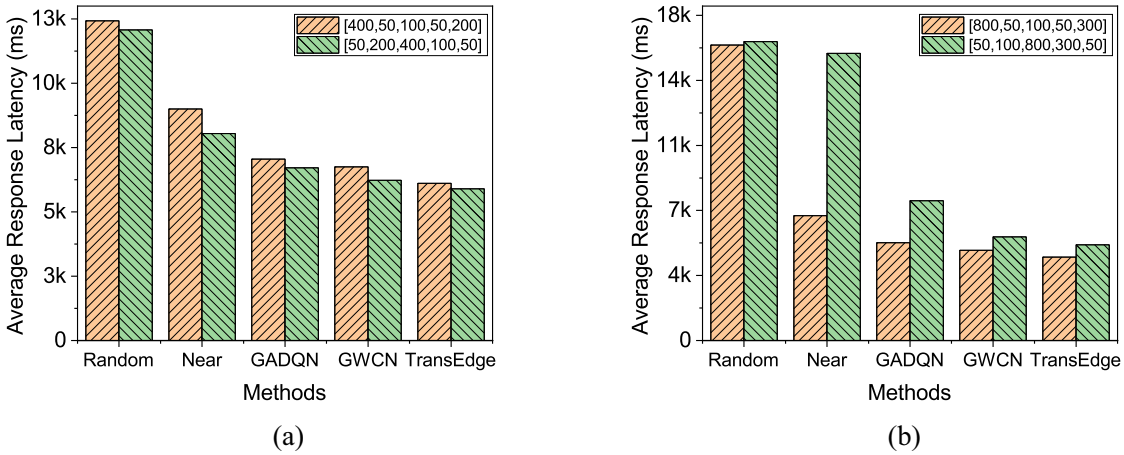

Fig. 14. Latency comparison of different algorithms under different resource placement. (a) Latency comparison on PEMS8. (b) Latency comparison on PEMS4.

to the other methods. Moreover, there are differences in the average response latency of all schemes under different resource placements. Specifically, the average response latency difference ranges from 2.8% to 10.6% on PEMS8 and 1.1% to 56.5% on PEMS4. The reason for this phenomenon may be that the performance of the server serving the sensor

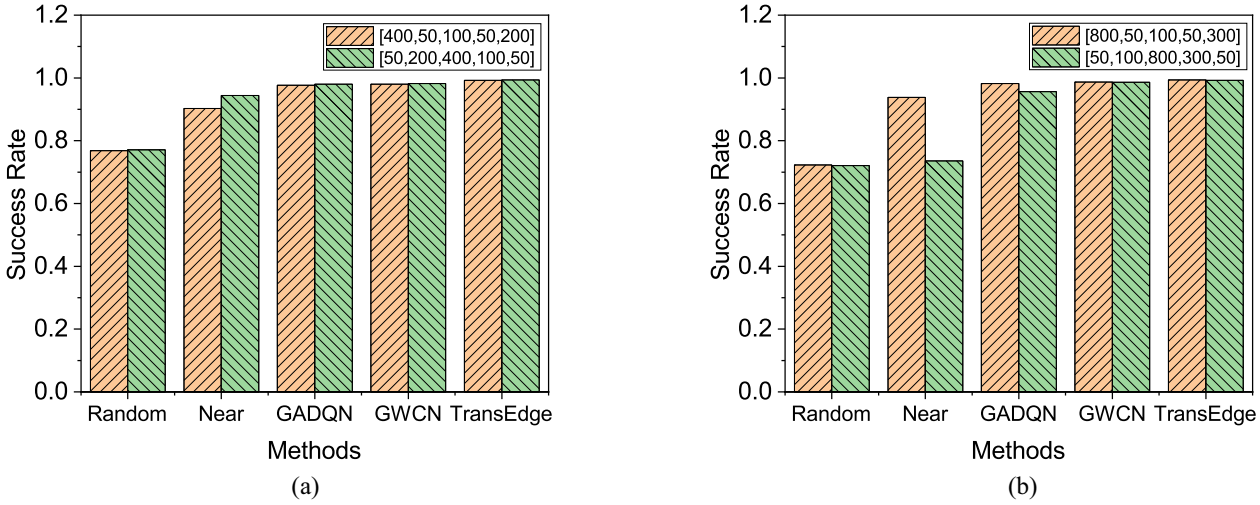

Fig. 15. Success rate comparison of different algorithms under different capacity. (a) Success rate comparison on PEMS8. (b) Success rate comparison on PEMS4.

has degraded, causing the tasks that originally needed to be processed on the current server to be frequently forwarded to other servers for processing, especially to the central cloud server, which implies a reduction in the success rate of the task request processing (as shown in Fig. 15), thereby increasing the response latency. Conversely, the response latency is reduced. This exploration is of great significance, and it promises to minimize the response latency through reasonable resource placement under the condition of limited resources.

VI. Conclusion

In this article, we propose a novel task offloading scheme based on GNNs and DRL in an edge-computing-enabled transportation system. Specifically, we first introduce a node placement scheme based on the AGA, aiming at allocating IoT sensors to appropriate edge servers to minimize TL. Then, we present an improved DRL scheme based on the GNN to learn the spatial features between sensors, aiming to obtain more reasonable task offloading decisions. Finally, a task forwarding strategy based on the greedy algorithm is designed to facilitate collaborative task offloading among different edge servers and overcome the system instability caused by the sudden surge of task requests. Extensive experimental results show that TransEdge reduces the response latency by at least 9.3% (on PEMS8) and 3.7% (on PEMS4) compared to four baseline algorithms while achieving a success rate of 99%. Interestingly, the average response latency of the same scheme under different resource placements differs by up to 56.5%, which implies that predicting the resource placement scheme of each edge server is crucial in the future.

References

[1] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibañez, "Internet of Vehicles: Architecture, protocols, and security," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3701–3709, Oct. 2018.

[2] S. Wan, R. Gu, T. Umer, K. Salah, and X. Xu, "Toward offloading Internet of Vehicles applications in 5G networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4151–4159, Jul. 2021.

[3] X. Xu, C. Yang, M. Bilal, W. Li, and H. Wang, "Computation offloading for energy and delay trade-offs with traffic flow prediction in edge computing-enabled IOV," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 15613–15623, Dec. 2023.

[4] Z. Nan, Y. Jia, Z. Ren, Z. Chen, and L. Liang, "Delay-aware content delivery with deep reinforcement learning in Internet of Vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 8918–8929, Jul. 2022.

[5] J. Lin, W. Yu, X. Yang, P. Zhao, H. Zhang, and W. Zhao, "An edge computing based public vehicle system for smart transportation," *IEEE Trans. Veh. Technol.*, vol. 69, no. 11, pp. 12635–12651, Nov. 2020.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[7] Z. Zhou, M. Shojafar, J. Abawajy, H. Yin, and H. Lu, "ECMS: An edge intelligent energy efficient model in mobile edge computing," *IEEE Trans. Green Commun. Netw.*, vol. 6, no. 1, pp. 238–247, Mar. 2022.

[8] K. Li, "Non-clairvoyant and randomised online task offloading in mobile edge computing," *Int. J. Parallel, Emerg. Distrib. Syst.*, vol. 37, no. 4, pp. 413–424, 2022.

[9] Q. He et al., "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.

[10] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May 2019.

[11] Z. Xiao, J. Shu, H. Jiang, G. Min, H. Chen, and Z. Han, "Perception task offloading with collaborative computation for autonomous driving," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 457–473, Feb. 2022.

[12] M. Z. Alam and A. Jamalipour, "Multi-agent DRL-based Hungarian algorithm (MADRLHA) for task offloading in multi-access edge computing Internet of Vehicles (IoVs)," *IEEE Trans. Wireless Commun.*, vol. 21, no. 9, pp. 7641–7652, Sep. 2022.

[13] Z. Chen, W. Yi, A. S. Alam, and A. Nallanathan, "Dynamic task software caching-assisted computation offloading for multi-access edge computing," *IEEE Trans. Commun.*, vol. 70, no. 10, pp. 6950–6965, Oct. 2022.

[14] X. Liu, J. Yu, Z. Feng, and Y. Gao, "Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing," *China Commun.*, vol. 17, no. 9, pp. 220–236, Sep. 2020.

[15] Z. Chen, W. Yi, A. Nallanathan, and J. Chambers, "Distributed digital twin migration in multi-tier computing systems," *IEEE J. Sel. Topics Signal Process.*, vol. 18, no. 1, pp. 109–123, Jan. 2024.

[16] Z. Fang, X. Xu, F. Dai, L. Qi, X. Zhang, and W. Dou, "Computation offloading and content caching with traffic flow prediction for Internet of Vehicles in edge computing," in *Proc. IEEE Int. Conf. Web Serv. (ICWS)*, 2020, pp. 380–388.

[17] T. Liu et al., "{BGL}:{GPU-efficient}{GNN} training by optimizing graph data {i/O} and preprocessing," in *Proc. 20th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2023, pp. 103–118.

[18] H. Zhang, H. Wang, Y. Li, K. Long, and A. Nallanathan, "DRL-driven dynamic resource allocation for task-oriented semantic communication," *IEEE Trans. Commun.*, vol. 71, no. 7, pp. 3992–4004, Jul. 2023.
[19] L. Liu, X. Yuan, N. Zhang, D. Chen, K. Yu, and A. Taherkordi, "Joint computation offloading and data caching in multi-access edge computing enabled Internet of Vehicles," *IEEE Trans. Veh. Technol.*, vol. 72, no. 11, pp. 14939–14954, Nov. 2023.
[20] Z. Ning et al., "Intelligent edge computing in Internet of Vehicles: A joint computation offloading and caching solution," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2212–2225, Apr. 2021.
[21] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.
[22] 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," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7519–7537, Nov. 2021.
[23] W. Fan, Z. Chen, Y. Su, F. Wu, B. Tang, and Y. Liu, "Accuracy-based task offloading and resource allocation for edge intelligence in IoT," *IEEE Wireless Commun. Lett.*, vol. 11, no. 2, pp. 371–375, Feb. 2022.
[24] B. Yang, X. Cao, X. Li, Q. Zhang, and L. Qian, "Mobile-edge-computing-based hierarchical machine learning tasks distribution for IIoT," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 2169–2180, Mar. 2020.
[25] M. S. Bute, P. Fan, L. Zhang, and F. Abbas, "An efficient distributed task offloading scheme for vehicular edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13149–13161, Dec. 2021.
[26] S.-S. Lee and S. Lee, "Resource allocation for vehicular fog computing using reinforcement learning combined with heuristic information," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10450–10464, Oct. 2020.
[27] X. Qi, J. Chong, Q. Zhang, and Z. Yang, "Collaborative computation offloading in the multi-UAV fleeted mobile edge computing network via connected dominating set," *IEEE Trans. Veh. Technol.*, vol. 71, no. 10, pp. 10832–10848, Oct. 2022.
[28] H. Guo, Y. Wang, J. Liu, and C. Liu, "Multi-UAV cooperative task offloading and resource allocation in 5G advanced and beyond," *IEEE Trans. Wireless Commun.*, vol. 23, no. 1, pp. 347–359, Jan. 2024.
[29] L.-H. Yen, J.-C. Hu, Y.-D. Lin, and B. Kar, "Decentralized configuration protocols for low-cost offloading from multiple edges to multiple vehicular fogs," *IEEE Trans. Veh. Technol.*, vol. 70, no. 1, pp. 872–885, Jan. 2021.
[30] X. Pu et al., "Incentive mechanism and resource allocation for collaborative task offloading in energy-efficient mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 72, no. 10, pp. 13775–13780, Oct. 2023.
[31] B. Liu, W. Zhang, W. Chen, H. Huang, and S. Guo, "Online computation offloading and traffic routing for UAV swarms in edge-cloud computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 8, pp. 8777–8791, Aug. 2020.
[32] Z. Ma, M. Xiao, Y. Xiao, Z. Pang, H. V. Poor, and B. Vucetic, "High-reliability and low-latency wireless communication for Internet of Things: Challenges, fundamentals, and enabling technologies," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7946–7970, Oct. 2019.
[33] A. Mahmood et al., "Industrial IoT in 5G-and-beyond networks: Vision, architecture, and design trends," *IEEE Trans. Ind. Informat.*, vol. 18, no. 6, pp. 4122–4137, Jun. 2022.
[34] W. Fan, S. Li, J. Liu, Y. Su, F. Wu, and Y. Liu, "Joint task offloading and resource allocation for accuracy-aware machine-learning-based IIoT applications," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3305–3321, Feb. 2023.
[35] L. Liu, M. Zhao, M. Yu, M. A. Jan, D. Lan, and A. Taherkordi, "Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 2, pp. 2169–2182, Feb. 2023.
[36] Z. Geng, X. Li, J. Wang, X. Li, Y. Zhang, and F. Wu, "A deep instance generative framework for MILP solvers under limited data availability," in *Proc. 37th Conf. Neural Inf. Process. Syst.*, 2024, pp. 1–23.
[37] V. Tiwari, C. Pandey, A. Dahal, D. S. Roy, and U. Fiore, "A knapsack-based metaheuristic for edge server placement in 5G networks with heterogeneous edge capacities," *Future Gener. Comput. Syst.*, vol. 153, pp. 222–233, Apr. 2024.
[38] K. Zhou, S.-K. Oh, J. Qiu, W. Pedrycz, K. Seo, and J. H. Yoon, "Design of hierarchical neural networks using deep LSTM and self-organizing dynamical fuzzy-neural network architecture," *IEEE Trans. Fuzzy Syst.*, vol. 32, no. 5, pp. 2915–2929, May 2024.
[39] S. Munikoti, D. Agarwal, L. Das, M. Halappanavar, and B. Natarajan, "Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, May 25, 2023, doi: 10.1109/TNNLS.2023.3283523.
[40] E. Nikishin et al., "Deep reinforcement learning with plasticity injection," in *Proc. 37th Conf. Neural Inf. Process. Syst.*, 2024, pp. 1–18.
[41] A. K. Jaiswal, S. Liu, T. Chen, Y. Ding, and Z. Wang, "Graph ladling: Shockingly simple parallel GNN training without intermediate communication," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 14679–14690.
[42] K. Huang, Y. Jin, E. Candes, and J. Leskovec, "Uncertainty quantification over graph with conformalized graph neural networks," in *Proc. 37th Conf. Neural Inf. Process. Syst.*, 2024, pp. 1–23.
[43] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 6, pp. 3425–3443, Jun. 2023.
[44] G. Wu et al., "Combining Lyapunov optimization with actor–critic networks for privacy-aware IIoT computation offloading," *IEEE Internet Things J.*, vol. 11, no. 10, pp. 17437–17452, May 2024.
[45] A. Alacaoglu, L. Viano, N. He, and V. Cevher, "A natural actor–critic framework for zero-sum Markov games," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 307–366.
[46] "TransEdge." 2024. [Online]. Available: https://github.com/xuaikun/TransEdge.git
[47] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 922–929.
[48] D. Chistikov, M. Englert, and R. Lazic, "Learning a neuron by a shallow ReLU network: Dynamics and implicit bias for correlated inputs," in *Proc. 37th Conf. Neural Inf. Process. Syst.*, 2024, pp. 1–13.
[49] W.-C. Chang and P.-C. Wang, "Adaptive replication for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 11, pp. 2422–2432, Nov. 2018.
[50] L. Kong et al., "Edge-computing-driven Internet of Things: A survey," *ACM Comput. Surveys*, vol. 55, no. 8, pp. 1–41, 2022.

Aikun Xu received the M.S. degree from Central South University, Changsha, China, in 2022, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering.

His research interests include deep learning, graph neural network, deep reinforcement learning, scheduling, electric vehicles, and edge computing.

Zhigang Hu received the B.S., M.S., and Ph.D. degrees from Central South University (CSU), Changsha, China, in 1985, 1988, and 2002, respectively.

In 2002, he joined CSU, where he is a Professor with the School of Computer Science and Engineering. He has published over 200 research papers. His research interests include radar signal processing and classification/recognition, high-performance computing, and cloud computing.

Xi Li received the M.S. and Ph.D. degrees in computer science from Central South University, Changsha, China, in 2005 and 2011, respectively.

She is currently an Associate Professor with the School of Computer Science and Engineering, Central South University. Her research interests include distributed systems and cloud computing, machine learning, digital twin, and intelligent manufacturing.

Rongti Tian received the M.S. degree from Central South University, Changsha, China, in 2023.

His research interests include deep learning, graph neural network, deep reinforcement learning, scheduling, electric vehicles, and edge computing.

Xinyu Zhang received the master's degree from Central South University, Changsha, China, in 2017, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering.

His main research interests include edge computing, cloud computing, and federated deep reinforcement learning.

Bolei Chen received the B.S. degree from the School of Computer Science and Engineering, South Central University for Nationalities, Wuhan, China, in 2020. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Central South University, Changsha, China.

His research interests include deep reinforcement learning, robotic navigation, and autonomous exploration.

Hui Xiao received the B.E. degree from Shandong University, Jinan, China, in 2017, and the M.E. degree from Central South University, Changsha, China, in 2020, where she is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering.

Her main research interests are in the area of mobile-edge computing and cloud computing.

Hao Zheng (Graduate Student Member, IEEE) received the M.S. degree from Central South University, Changsha, China, in 2021, where he is currently pursuing the Ph.D. degree in the research group of Zhigang Hu with the School of Computer Science and Engineering.

His research interests include synthetic aperture radar image processing, transfer learning, and computer vision.

Xianting Feng received the B.S. degree in 2022. She is currently pursuing the M.S. degree with the Department of Computer Science and Engineering, Central South University, Changsha, China.

Her research interests include synthetic aperture radar image processing, transfer learning, and computer vision.

Meiguang Zheng received the B.S. and Ph.D. degrees in computer science from Central South University, Changsha, China, in 2005 and 2011, respectively.

She is currently an Associate Professor with the School of Computer Science and Engineering, Central South University. She is currently leading some research projects supported by the National Natural Science Foundation of China. Her research interests include federated learning, distributed machine learning, computer vision, and edge computing.

Ping Zhong (Member, IEEE) received the Ph.D. degree in communication engineering from Xiamen University, Xiamen, China, in 2011.

She is currently an Associate Professor with the School of Computer Science and Engineering, Central South University, Changsha, China. Her research interests include machine learning, data mining, and network protocol design.

Dr. Zhong is a member of ACM, CCF, and IEICE.

Keqin Li (Fellow, IEEE) is currently a SUNY Distinguished Professor of Computer Science with the State University of New York, New Paltz, NY, USA. He has published over 620 journal articles, book chapters, and refereed conference papers. His current research interests include cloud computing, fog computing, mobile-edge computing, energy-efficient computing and communication, embedded systems, cyber–physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU–GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, and intelligent and soft computing.

Prof. Li received several best paper awards. He currently serves or has served on the editorial boards of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON SERVICES COMPUTING, and IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING.