Resource Aware Multi-User Task Offloading In Mobile Edge Computing

Shunhui Ji[1,2], Jiajia Li[1,2], Huiying Jin[3], Hai Dong[4], Zhiyuan Ge[1,2], Shuhan Yang[1,2], and Pengcheng Zhang[1,2,*]

[1]Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, Nanjing, China
[2]College of Computer Science and Software Engineering, Hohai University, Nanjing, China
[3]School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China
[4]School of Computing Technologies, RMIT University, Melbourne, Australia
{*shunhuiji, jiajiali, zhiyuange, pchzhang*}*@hhu.edu.cn, hyjin@njupt.edu.cn, hai.dong@rmit.edu.au, 1114168847@qq.com*

***Abstract*—Mobile edge computing (MEC) relies on offloading tasks to edge nodes to avoid delays and failures caused by local computing. However, developing efficient offloading decisions is challenging, as it involves addressing the intricacies of tasks and the instability of edge node resources(e.g. available computer resources, memory, and bandwidth). In this paper, we propose a novel approach to tackle the problem of task offloading. Our approach involves dividing tasks into smaller units and considering the correlations between these sub-tasks. To make optimal offloading decisions, we employ a deep reinforcement learning algorithm that takes into account user movement patterns and the availability of resources at edge nodes. Through simulations, we demonstrate that our proposed algorithm outperforms several existing algorithms in terms of offloading decisions. It effectively reduces task execution delays and energy costs. These findings highlight the potential of our approach in improving the performance of task offloading in MEC systems.**

***Index Terms*—Mobile Edge Computing, Task Offloading, Trajectory prediction, Resource Aware, Deep Reinforcement Learning**

I. Introduction

As Internet of Things (IoT) applications continue to advance, users are generating increasingly computationally intensive tasks when running applications such as online games, virtual reality/augmented reality, and autonomous driving [1]. These tasks often require low delay. However, mobile devices are limited by their local resources, which restricts their ability to meet the computational resource and power consumption requirements of such tasks. To address this issue, Mobile Edge Computing (MEC) has emerged as a novel computing paradigm [2]. MEC leverages the computing capabilities of edge nodes (ENs) to enable edge users (EUs) to offload their computational tasks to nearby ENs for processing. This offloading reduces task processing delay and lowers local task execution power consumption [3]. However, in the realm of MEC, the complexity of tasks and the dynamic nature of environmental conditions pose significant challenges in determining optimal offloading decisions.

Considerable research has been conducted on the subject of computation offloading. Previous studies primarily focused on offloading computational tasks to edge servers for processing within fixed time intervals, statically allocating resources in the servers to these offloaded tasks, neglecting the continuous and dynamic nature of the task offloading problem across multiple moments in time. Chen et al. [4] assumed that ENs provide EUs with sufficient resources to meet task resource requirements and conducted research on the computation offloading problem to minimize task processing delay and device energy consumption. However, in real-life scenarios, different ENs possess distinct resource capacities, such as CPU frequency, memory size, and bandwidth. The available resources in edge servers dynamically adjust with changing loads. Therefore, in the process of making task offloading decisions, it is imperative to consider the dynamic and finite nature of edge node resources.

Recent studies have been investigating dynamic task offloading in MEC systems. Existing research utilizes reinforcement learning to address the task-offloading problem in dynamic environments [5]–[8]. Algorithms based on reinforcement learning enable EUs to continuously update offloading strategies by interacting with the environment without needing to know other users' strategies. These studies tend to treat tasks generated by edge users as independent tasks for offloading decisions. However, this approach cannot be applied to large, complex, and high-computational-density task offloading scenarios. In vehicular edge computing (VEC), some studies [9], [10] consider task dependencies among vehicles. They predict vehicle movement trajectories to detect inter-vehicle communication times, determine task priorities, and optimize system-wide task completion delay. However, these methods still face the following issues in the process of generating task-offloading strategies:

- *The dynamic fluctuation in resource availability experienced during the mobility of edge users often leads to delayed offloading decisions.* In real-world situations involving multi-user task offloading, mobile users are required to make decisions regarding task offloading while in motion. As the EU moves, the EN resources are changing dynamically, and the EU may move out

* Corresponding Author.

S. Ji et al., "Resource Aware Multi-User Task Offloading In Mobile Edge Computing," 2024 IEEE International Conference on Web Services (ICWS), Shenzhen, China, 2024, pp. 665-675, doi: 10.1109/ICWS62655.2024.00086.

of the node coverage, which leads to an increase in the cost of task execution or even task execution failure. Nevertheless, current offloading solutions tend to rely solely on the present resource status for decision-making. This approach overlooks the dynamic nature of available resources at edge nodes, potentially resulting in delays in decision-making processes. Hence, it is imperative to incorporate the anticipation of future resource availability into offloading decisions to ensure adaptability to users' mobility scenarios.

- *The complexity of offloading decisions is influenced by the characteristics of sub-task data and their dependencies.* Although some existing studies [9]–[11] have considered the dependency relationships between tasks, they only focus on the data dependency characteristics of tasks to optimize the data transmission delay between tasks, without fully considering the impact of task dependency relationships on offloading decisions. Due to the strict sequential constraints on task execution, the offloading decision of subsequent tasks depends on the offloading decision of preceding tasks. Therefore, in handling complex associated task-offloading processes, it is necessary to consider the actual task-offloading situation to formulate globally optimal offloading decisions, thereby minimizing the total cost of task execution and maintaining system stability.

This paper proposes a multi-user task offloading strategy algorithm called LSTM-MADDPG (Long Short Term Memory Multi-Agent Deep Deterministic Policy Gradient) to address the issues related to dynamic task offloading decision-making for EUs during their movement. The algorithm has two main components. Firstly, it uses LSTM models to predict the movement trajectories of EUs and to perceive the dynamically available resources when EUs access ENs. Secondly, it employs the MADDPG algorithm to formulate a fine-grained task offloading strategy for multiple users. The algorithm considers the dependencies between tasks as a directed acyclic graph, taking into account the interrelatedness of tasks and incorporating these relationships when making offloading decisions. Additionally, the algorithm fully considers the competition among multiple users for limited resources in the edge environment. The ultimate objective of the algorithm is to minimize the overall cost of global task execution. The main contributions are as follows:

- We propose an algorithm named LSTM-MADDPG that considers the movement trajectories of EUs to tackle the task offloading challenge when these users are in motion. By predicting the future locations of users and evaluating the resources at edge nodes, our objective is to enable real-time task offloading decisions for multiple users during their mobility.
- We consider the dependencies between sub-tasks as a Directed Acyclic Graph (DAG), with task data features and dependencies treated as state information. We utilize deep reinforcement learning algorithms to determine the optimal execution nodes for sub-tasks, achieving fine-grained optimal task offloading decisions.
- This paper assesses the performance of the algorithm through simulation experiments. The results indicate the effectiveness of the algorithm by comparing it with various other task-offloading decision algorithms. The results confirm that the proposed algorithm is capable of reducing the overall cost of user task execution.

The rest of this paper is organized as follows: Section II introduces the related work. Section III describes the details of the task offloading problem model in MEC. Section IV describes the proposed task offloading method. Section V presents the experimental evaluation. Finally Section VI concludes the paper.

II. Related Work

Existing research on task offloading optimization methods can be separated into mathematical model-based methods and reinforcement learning-based methods. To better understand the limitations of the existing studies, we investigated existing relevant task offloading schemes. Table I displays the objectives, user mobility, task dependency, and primary algorithms of each method.

A. Mathematical models-based method

Zhang et al. [12] formulated the task offloading problem as a Multi-Stage Stochastic Programming (MSSP), aiming to minimize the overall delay of offloading. Bi et al. [13] proposed a joint optimization method based on the Alternating Direction Method of Multipliers (ADMM) decomposition technique. The method aims to maximize the overall task execution rate by optimizing the computation mode selection for individual users. As more users join the network, the algorithm complexity increases. T. X. Tran et al. [14] formulated the joint problem of task offloading and resource allocation as Mixed-Integer Nonlinear Programming (MINLP), utilizing convex optimization theory to optimize task offloading decisions. They proposed a heuristic algorithm to address this problem. Tang et al. [15] introduced a caching strategy and fine-grained task migration strategy on edge clouds based on a Genetic Algorithm (GA). They aim to devise an optimal strategy for mobile device task migration, thereby reducing energy consumption in multi-access edge computing. However, in a dynamic multi-access edge computing environment, obtaining approximate solutions using heuristic methods can be a time-intensive and memory-consuming process. Wu et al. [16] proposed an adaptive offloading decision algorithm based on Lyapunov optimization, aiming to minimize overall energy consumption. In summary, traditional mathematical model optimization algorithms require precise models and significant computational time, which makes them unsuitable for dynamic MEC environment configurations.

B. Reinforcement learning-based methods

With the development of artificial intelligence, reinforcement learning has been applied to MEC task offloading.

TABLE I
A COMPARISON OF RELATED STUDIES.

Reference	Objectives	Mobility	Dependency	Resource aware	Scheme
[4]	Delay,Energy				TADPG
[7]	Delay,Energy	✓			DDQN
[9]	Delay,Energy	✓	✓		MESON
[12]	Delay	✓			SAA-MSSP
[13]	Energy				ADMM
[14]	Delay,Energy	✓		✓	hJTORA
[15]	Energy	✓	✓		GA
[16]	Energy	✓		✓	LARAC
[17]	Delay	✓	✓		DRL
[18]	Delay,Energy	✓		✓	JCOTM
[19]	Delay	✓			COMA

Chen et al. [4] investigated the joint problem of task offloading and resource allocation and proposed a Time-Attention Deterministic Policy Gradient algorithm (TADPG) to optimize task completion's delay and energy consumption. This method assumes that EUs can obtain sufficient computing and transmission resources from ENs. However, the resources in ENs may be subject to capacity limitations. Wang et al. [17] introduced a DRL-based offloading framework that can automatically discover various scenarios, inferring optimal offloading strategies for different scenarios. However, this method requires more time and scenario data for training a general framework, resulting in insufficient adaptability to new scenarios. Wu et al. [18] proposed a DRL-based Joint Computing Offloading and Task Migration Optimization (JCOTM) algorithm, aiming to reduce system energy consumption and task delay. These methods only consider the existence of a single intelligent agent in the MEC environment. However, in cases where multiple intelligent agents are present in the same environment, each agent tends to rely solely on locally observed state information to make its own decisions. This leads to the potential for instability in the overall MEC, and the algorithm may even fail to converge.

Recent research has attempted to apply multi-agent reinforcement learning algorithms to MEC task offloading problems. Liu et al. [19] proposed a distributed task migration algorithm based on the Cooperative Multi-Agent (COMA) reinforcement learning method, aiming to address the task migration problem in distributed user scenarios. Tang et al. [7] introduced a decentralized computation offloading algorithm based on DQN suitable for mixed cooperative-competitive heterogeneous MEC environments, aiming to improve user task completion rates. Zhao et al. [9] developed a computation offloading model based on VEC, considering the data dependency of tasks.

The previous works have relied solely on the current state to make offloading decisions. However, since edge node resources are in a dynamically changing state due to the user movement process, offloading decisions based solely on the current resource status may have hysteresis. Moreover, in complex MEC environments, tasks are often interdependent, and the execution location of preceding tasks significantly impacts the execution cost of subsequent tasks. Therefore, we aim to design a new multi-user task offloading algorithm that can meet the current requirements.

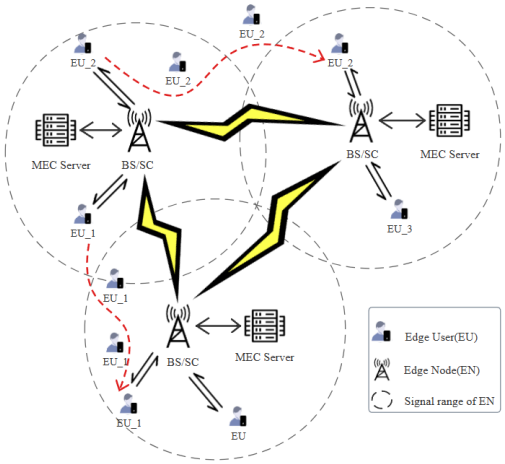

Fig. 1. MEC model of multi-user and multi-edge nodes.

III. SYSTEM MODEL

In this section, we first introduce the multi-user MEC system model. Next, we define the task model. Following that, we describe the delay and energy costs associated with task execution. Finally, we present the task offloading decision model.

A. MEC Model

As illustrated in Fig.1, the MEC system consists of D edge users and N edge nodes. To provide services to EUs, edge servers are deployed on edge nodes. EUs move about in the MEC environment. For example, as EU_1 moves between ENs, restricted local resources prevent the completion of computationally expensive tasks. Therefore, EU_1 can transfer locally generated tasks to ENs, allowing collaborative task completion. Each node contains servers with finite computational and storage capacities, indicated as $ENs = \{1, 2 \ldots, N\}$. EN_i is expressed as follows:

$$EN_i = (fr_i, mem_i, core_i, lat_i, lng_i) \tag{1}$$

fr_i, mem_i and $core_i$ represent the CPU computing frequency (GHz), memory capacity, and the number of cores of EN_i respectively. (lat_i, lng_i) denote the latitude and longitude coordinates of EN_i. The coverage range of a base station is typically 2-5 kilometers [20], and the signal is usually transmitted in rays. Based on this, we consider the coverage area of the base station signal to be circular with a radius of 3 kilometers.

The system defines D EUs, represented as $EUs = \{1, 2, ..., D\}$. EU_j is represented as follows:

$$EU_j = (fr_j, mem_j, core_j, EC_j, ET_j) \tag{2}$$

EC_j and ET_j respectively represent the computation energy consumption coefficient and transmission energy consumption coefficient of EU per unit of time. Due to the mobility of the EU, the EU's location coordinates may change. $(lat_j(t), lng_j(t))$ represents the coordinates of the user at time slot t. If the EU is inside the EN's coverage region, it can exchange data with the EN.

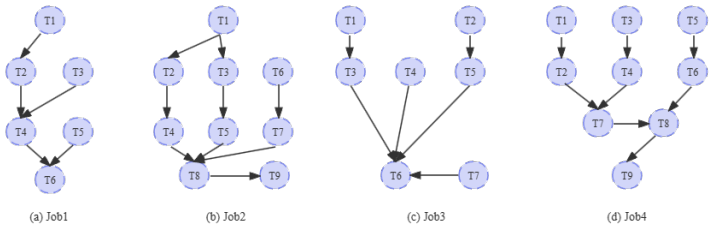

Fig. 2. The DAG structures for different jobs.

B. Task Model

We consider the computing workloads created by the EU as having a DAG network topology. As illustrated in Fig.2, Job(2) may be broken into nine sub-tasks $\{T_1, T_2, ..., T_9\}$. We consider each episode consisting of T time slots with equal intervals $\mathbb{T} = \{1, 2, ..., n\}$, and each time slot lasts for Δ_t seconds. We assume that, at the beginning of each time slot, the EU can generate local tasks, and the task generation probability follows a Poisson distribution [7]. This assumption is consistent with previous research.

At the beginning of time slot $t \in \mathbb{T}$, when EU $d \in D$ has a new task arrival, the job is represented as:

$$job_d^k = \{T_1, T_2, \ldots, T_n, R_k\} \tag{3}$$

$$R_k = \{r_i | r_i = (pre_{i,p}, succ_{i,s}, I(pre_{i,p}), O(succ_{i,s}))\} \tag{4}$$

R_k represents the relationship between sub-tasks, $pre_{i,p}$ and $succ_{i,s}$ respectively represent the predecessor and successor tasks of the i-th sub-task, simultaneously satisfy $O(pre_{i,p}) = I(succ_{p,i})$. $T_i = \{I_i, O_i, E_i\}$, I_i, O_i, and E_i respectively represent the input data size, output data size, and the number of instructions for T_i.

When a job arrives for EU $d \in D$, task offloading strategies must be determined by EU d based on the present observable MEC state information. This includes deciding whether a sub-task should be carried out locally or sent to an EN for execution via the signal channel. The detailed information observed by each EU will be discussed in the next section. There are two components to any EU task offloading decision: 1) Whether offloading is needed; 2) If so, to which EN the task should be offloaded. Therefore, we define $\rho_d(T_i)$ to represent whether the task is offloaded, where $\rho_d(T_i) = 0$ indicates local execution of the task, $\rho_d(T_i) = 1$ indicates offloading the task to an EN. Furthermore, $\theta_{d,n}(T_i)$ is defined to represent whether the task is offloaded to EN n, where $\theta_{d,n}(T_i) = 1$ indicates execution of the task on EN n, and conversely, $\theta_{d,n}(T_i) = 0$. Based on the above description, the decision actions made by each EU for each sub-task are as follows:

$$a_d(T_i) = \{\rho_d(T_i), \theta_d(T_i)\} \tag{5}$$

C. Cost Model

1) Cost of local computation: The local computation cost comprises both the delay and the energy consumption costs. For a task T_i executed locally, the delay primarily depends on the local device's CPU frequency $fr_d, d \in D$, representing the local computing capability. we define Que_d as the local execution queue. At time slot $t \in \mathbb{T}$, when the task is placed for local execution, it is added to the queue. In this work, we assume that tasks in the queue can be executed concurrently, with computing resources evenly distributed among them. Since tasks arrive at different times, the available resources for each task dynamically change. The number of concurrent tasks is represented by $len(Que_d)$. Based on these considerations, the delay cost of local task execution is defined as follows:

$$\mathbb{D}_d^{T_i} = \frac{(E_{T_i} * \mathcal{C})}{\mathcal{F}(fr_d, core_d)} \tag{6}$$

$\mathcal{F}(\cdot) = \frac{fr_d \cdot core_d}{len(Que_d)}$ represents the available computing resources for a single task, and $\mathcal{C}$ is a constant denoting the CPU cycles required to process a unit bit. It is notable that, when tasks are completed locally, the delay in input data transfer is not taken into account as in [21]. There is no transmission delay between tasks if the preceding task $pre_{i,p}$ is done locally. When the previous task $pre_{i,p}$ proceeds at the edge node, the output data often has a relatively small size, and the delay in returning data from EN to the device is not considered [22].

Next, the energy cost of local processing is defined as follows:

$$\mathbb{E}_d^{T_i} = \mathbb{D}_d^{T_i} * EC_d \tag{7}$$

EC_d (J/S) is the energy consumption of the EU per unit of time for computation. Based on the above description, the cost formula for task T_i executed locally is as follows:

$$Cost_d^{T_i} = \alpha * \mathbb{D}_d + \beta * \mathbb{E}_d^{T_i} \tag{8}$$

We consider delay and energy consumption as two components of the task execution cost, which are normalized and aggregated to form the total execution cost, similar to [8]. The delay and energy consumption weights are represented by α and β, with $\alpha + \beta = 1$. Only the local execution delay cost is considered when $\alpha = 1$ and $\beta = 0$. Conversely, just the local execution energy consumption cost is taken into account.

2) Cost of edge node computation: We determine the cost of offloading task T_i for execution on the EN. The cost of running task T_i on the edge node consists of delay and energy costs, similar to when tasks are executed locally. In contrast to the delay cost of local execution, when a task is done on EN, EU d first transmits the task to the edge node while also taking into account the output data transfer between tasks. The delay cost of sending output data of $pre_{i,p}$ to the edge node must be determined if $pre_{i,p}$ is carried out locally. The transmission delay of EU d involves two parts: 1)transmission input data delay and 2) transmission task instruction delay. The bandwidth and quality of the transmission link between the ENs and the EUs affect the transmission delay from EU d to EN n. We have considered a wireless communication model where edge users transmit data over orthogonal channels [23]. In the edge computing environment, wireless transmission experiences path loss and fading as the distance between EU and EN increases [24]. To simulate a real data transmission scenario, it is essential to consider the attenuation of data rates in wireless transmission. Thus, we define the transmission rate

from EU d to EN n denoted as $\mathbb{R}_d^n$(Mbps)that can be obtained through Shannon's theory [25], calculated as follows:

$$\mathbb{R}_d^n = \omega * \log_2 (1 + \frac{|\mathcal{H}_{d,n}| * \mathcal{P}}{\sigma^2 * (dist_d^n)^2}) \tag{9}$$

where ω represents the wireless transmission channel bandwidth from ED $d \in D$ to EN $n \in N$, $|\mathcal{H}_{d,n}|$ represents the wireless transmission channel wireless gain, $\mathcal{P}$ and σ represent the transmit power at the device side and the received noise power at the edge node, respectively. $dist_d^n$ is the distance between EU and EN. Since the coordinates of EU and EN are supplied in latitude and longitude, the distance between them is calculated using the Haversine formula, as indicated in the following equation:

$$dist_d^n = 2arcsind * R \tag{10}$$

$$d = \sqrt{sin(\frac{a}{2})^2 + cos(lat_d(t)) * cos(lat_n) * sin(\frac{b}{2})^2} \tag{11}$$

$$a = lat_d(t) - lat_n \tag{12}$$

$$b = lng_d(t) - lng_n \tag{13}$$

where $R = 6378.137$. We calculate the actual bandwidth between EU d and EN based on the distance between EU d and EN at time slot t.

According to the above wireless transmission model, we define the cost of task transmission delay as follows:

$$\mathbb{D}_n^{T_i} = \frac{\sum I(pre_{i,p}) + E_i}{\mathbb{R}_d^n}, \text{where } \rho_d(pre_{i,p}) = 0 \tag{14}$$

In addition, based on the transmission delay, we obtain the device's transmission energy consumption, calculated as follows:

$$\mathbb{E}_n^{T_i} = \mathbb{D}_n^{T_i} * ET_d \tag{15}$$

ET_d (J/S) is the energy consumption of the EU per unit of time for transmission.

The task's instruction size and the amount of computing capacity that is now available on the offloaded edge node define the computational delay cost. Similar to the local execution process, the EN $n \in N$ also maintains a task queue Que_n, evenly distributing the available resources of the edge node. Therefore, the EN n task computation delay is calculated as follows:

$$\hat{\mathbb{D}}_n^{T_i} = \frac{E_i * \mathcal{C}}{\mathcal{F}(avail_{fr_n(t)}, core_n)} \tag{16}$$

$avail_{fr_n(t)}$ represents the available computing resources of EN n at time slot t. Since the standby power consumption of EU is relatively low when the task is executed on EN, we do not consider device power consumption here. Therefore, the total cost of offloading task T_i to be executed on edge node n is calculated as follows:

$$Cost_n^{T_i} = \alpha * (\mathbb{D}_n^{T_i} + \hat{\mathbb{D}}_n^{T_i}) + \beta * \mathbb{E}_n^{T_i} \tag{17}$$

D. Task Offloading Model

As previously indicated, in a multi-user mobile edge environment, coordinating the offloading decisions of multiple users is crucial to minimize the overall task execution costs globally. Therefore, we formulate the task offloading problem in a multi-user mobile edge environment as a multi-agent resource competition problem. The total system cost is expressed as the sum of the task execution delay cost and the energy consumption cost for multiple users. The objective is to minimize the overall system cost while satisfying resource constraints in the MEC system. The specific formulation of the optimization problem is as follows:

$$\mathbb{P}_1 : \min \sum_{d}^{D} \sum_{T_i}^{K} \rho_d(T_i) * Cost_d^{T_i} + (1 - \rho_d(T_i)) * Cost_d^{T_i} \tag{18}$$

$$\text{s.t.} \quad C_1 : \rho_d(T_i) = 0, 1, \forall d \in D \tag{18a}$$

$$C_2 : \theta_d^n(T_i) = 0, 1, \forall d \in D \tag{18b}$$

$$C_3 : \sum_{n}^{N} \theta_d^n(T_i) \leq 1 \tag{18c}$$

$$C_4 : \sum_{d}^{D} fr_n(T_i) \leq fr_n \tag{18d}$$

$$C_5 : \sum_{d}^{D} mem_n(T_i) \leq mem_n \tag{18e}$$

C_1, C_2, and C_3 specify that each EU has only two choices for processing each sub-task: locally or offloading to an edge node. That is, each EU can either offload a sub-task or keep it locally. Additionally, each sub-task is offloaded to exactly one edge node. Constraints C_4 and C_5 ensure that the tasks offloaded by EU to EN do not exceed the total computing and memory resource capacity of the EN.

IV. Multi-Agent Task Offloading Algorithm

Task offloading has been indicated to be an NP-hard issue in the literature [26]. To address this problem, we propose a Reinforcement Learning-based resource-aware task offloading method. The method is detailed in Algorithm IV. Its main framework is shown in Fig. 3. There are three primary components to this method:

Data collection and preprocessing. To evaluate task offloading decisions when edge users are moving, we use current datasets to create a dataset that meets the requirements of a mobile scenario. Firstly, the EU trajectory dataset must be filtered to exclude data points with a state value of 0 (the EU is stationary) and duplicate location points. Secondly, to guarantee that the experimental data closely mimics true resource utilization circumstances, it is critical to filter away incorrect data, e.g. samples with CPU usage equal to zero.

Perceiving available resources. Firstly, when a new task arrives at the EU, we extract the DAG information of the task(see line [15] in Algorithm 1). In addition, we constructed

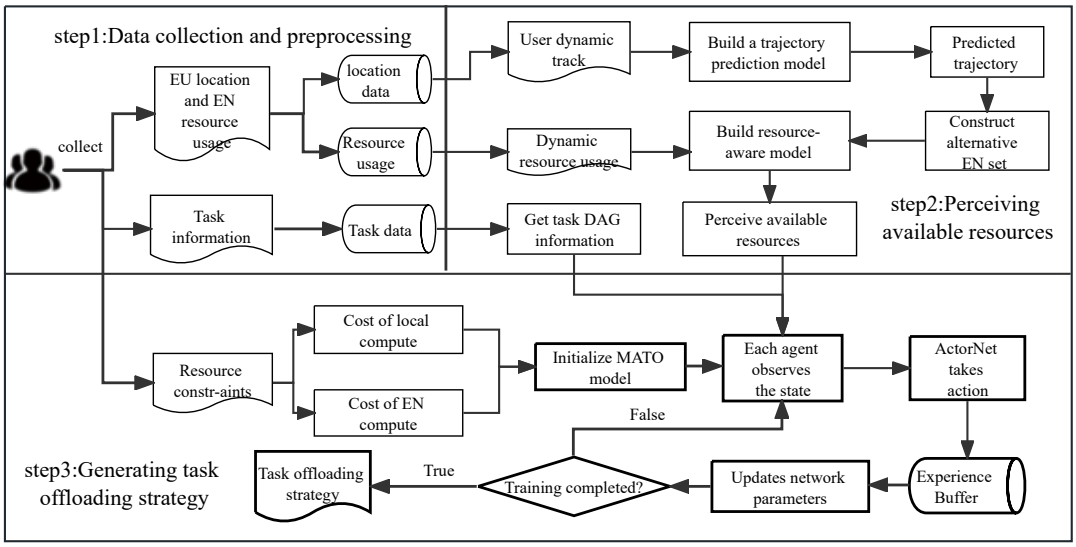

Fig. 3. Resource-aware multi-user task offloading method framework.

a resource-aware model to predict the available resources of alternative edge nodes. The trajectory prediction model is built using movement trajectory data from the EU. The next position of the EU may be predicted using the learned trajectory prediction model. The distance between the expected position and the ENs is used to calculate the alternative edge nodes for task offloading(see lines [16-17] in Algorithm 1). Following that, based on the previous resource consumption patterns in the alternative edge nodes, a resource prediction model is employed in estimating the edge nodes' future available resource information(e.g. CPU availability and accessible memory capacity) (see line [18] in Algorithm 1). The association between the moveable trajectory of the EU and the resources that are accessible to the EN may be determined by the use of the resource prediction model.

Generating task offloading strategy. Based on resource restrictions, create a local computing cost model and an edge node computing cost model, then initialize the Multi-Agent Task Offloading (MATO) network (see lines [1-4] in Algorithm 1). Each EU is treated as an agent, generating offloading actions based on the observed environmental state (see lines [6-13] in Algorithm 1). The EU interacts with the environment based on its actions and receives action rewards(see line [14] in Algorithm 1). The EU strategies and rewards are stored in an experienced pool for replay to update network parameters (see line [20] in Algorithm 1). This approach allows the EU to generate offloading decisions (Action) in a decentralized fashion based on observable MEC status. Actions are reviewed using a centralized assessment technique to improve policy network stability and raise the incentive value for offloading decisions(see lines [21-26] in Algorithm 1).

A. Data collection and preprocessing

Firstly, user movements during the day are documented in the EU's mobile trajectory collection. We concentrate on the mobility process of the user, eliminating duplicate samples from the dataset and filtering out cases when the user's status is 0 (signaling a stationary state). To reflect realistic edge node usage, we then remove nodes with resource consumption equal

Algorithm 1 Multi-Agents Task Offloading Algorithm

Input: Trajectory of EU, Resource utilization of EN, Task information

Output: Offload decision

1: Initialize N Edge Nodes and D Edge Users Parameters
2: Initialize Buffer $memory$ and Assessment Threshold $Evaluate$
3: Setting the Random $seed$
4: Set the initial exploration rate $\epsilon = 1$
5: **for** each $episode = 1, 2, \ldots, E$ **do**
6: Agent gets observation state: $\mathcal{S} = \{s_1(0), s_2(0), \ldots, s_d(0)\}, d \in D$
7: **for** each $t \in \mathbb{T}$ **do**
8: Select a arbitrary probability value $e \in [0, 1]$
9: **if** $e \leq \epsilon$ **then**
10: Choose action: a_d=ActionNet($s_d(t)$)+OUNoise
11: **else**
12: Choose action: a_d=ActionNet($s_d(t)$)
13: **end if**
14: Receive the Reward r from the environment
15: New job arrival d: $Job_d(t+1)$
16: Predicting $H(t+1)$ based on historical locations
17: Get candidate servers: $\mathbb{P}_d(t+1) = \{EN_1, \ldots, EN_q\}$
18: Alternative EN Resource Awareness: $\mathbb{H}_d(t+1)$
19: Observe the next state:$s_d(t+1)$
20: Store the experience$\{s_1(t), \ldots., s_d(t), a_1, \ldots, a_d, r_1, \ldots, r_d, s_1(t+1), \ldots., s_d(t+1)\}$into $memory$
21: **for** d=1,2,...,D **do**
22: **if** Number of samples $\geq Evaluate$ **then**
23: Sample a mini-batch of experiences from $memory$
24: Update critic-network and actor-network
25: Soft update the target network parameters
26: **end if**
27: **end for**
28: **end for**
29: **end for**

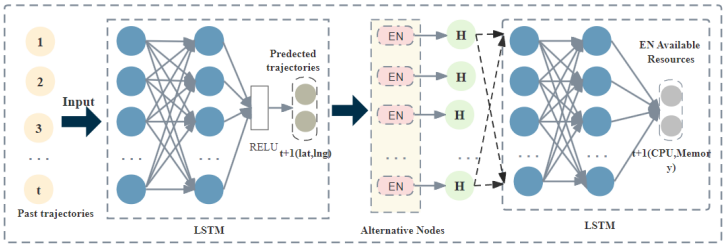

Fig. 4. Awareness of available resources model network structure.

to 0 (which indicates that the EU has not reached the node) from the dataset about edge node resource utilization. We use actual task composition data from Dataset 3[1], as seen in Fig.2, and use this structure to represent the task's DAG produced by the EU to replicate real-world task decomposition.

B. Perceiving available resources

To achieve awareness of the available resources on edge nodes, we have employed the LSTM model. The LSTM is a popular technique for learning temporal relationships in continuous observations and predicting future changes in time series [27]. LSTM can efficiently capture long-term dependencies and temporal patterns in such data, as changes in edge node resources and user trajectories show temporal relationships. This capability makes it an ideal choice for handling data with temporal characteristics.

To begin, we train a trajectory prediction model using the acquired historical trajectory information of EUs, to predict the future coordinates of the EU. The input layer of the trajectory prediction model is responsible for taking the EU's historical trajectory as input and passing it on to the subsequent layers. The representation of the EU's historical trajectory is denoted as $H(t) = (lat_t, lng_t)$. By inputting historical trajectory data into the model, the prediction for the EU's location at the next moment, $H(t+1)$ is determined. The trajectory prediction model is represented as:

$$H(t+1) = \mathbb{F}(H(t), H(t-1), \dots H(1)) \tag{19}$$

Specifically, the trajectory prediction model takes the matrix H as input to learn the EU's historical trajectory characteristics. The trajectory prediction model has two hidden layers and one activation layer with 2 hidden outputs for 2 columns. As shown on the left side of Fig.4, this model contains $2t$ LSTM units. Each LSTM unit takes a time trajectory $H(t)$ as input, these LSTM units are fully connected in sequence to track the sequence changes from $H(1)$ to $H(t)$ and capture user trajectory characteristics. The LSTM network will output information indicating that all historical trajectories are included in the last LSTM unit. This output will be passed to the activation layer for further learning. Finally, the predicted two-dimensional trajectory coordinates $H(t+1)$ will be output.

Next, we calculate the distance between users' expected future coordinates and ENs to determine if they are inside the coverage range of edge nodes. This procedure aids in selecting the set of candidate offloading nodes. Similarly, we use nodes' past utilization of resource data to train a resource-aware model. The resource-aware model has 2 hidden layers and 1 output layer with an output dimension of 2. As shown on the right side of Fig.4, the trained model utilizes the historical records of edge node resource usage to predict the future resource utilization levels of edge nodes soon.

We utilize the Haversine formula to determine the distance $dist_d^n(t+1)$ between the EU and the EN based on their position at time slot $t+1$. The pre-allocated candidate edge node set is then created using this distance and the edge node's coverage range, where $\mathbb{P}_{t+1}^d = \{EN_1, \dots, EN_q\}$. Based on the candidate edge nodes for edge users, we apply the resource-awareness model to forecast the resource use of alternative edge nodes. The historical use of edge nodes is denoted by $\mathbb{H}(t) = (CPU_t, Mem_t)$. $\mathbb{H}(t+1)$ represents the resource usage status of the edge node at the next moment. It should be noted that the resource-awareness model is used to determine the alternative edge nodes' resource utilization status. Edge nodes that are not included in the candidate set are deemed unreachable by the edge user, therefore their available resources are presumed to be 0.

C. Multi-agent task offloading algorithm

We apply the Multi-Agent Deep Deterministic Policy Gradients (MADDPG) algorithm [26] to make multi-user task offloading decisions. Each EU is regarded as an agent in the environment, with competition for resources among them. Every agent sees its current state data at the start of every slot, which includes the DAG of the task, position data, alternative node-set, and available resource information. The agent makes decisions about offloading local tasks based on this state of knowledge. The method uses a distributed execution strategy with centralized training. Based on partially visible state information, the action network of each agent develops its own decisions, continuously updating these decisions. The centralized evaluation network draws samples from the experience pool for evaluation and updates its parameters accordingly. The ultimate goal of this algorithm is to minimize the overall execution cost of user tasks by optimizing the mapping from state to policy.

State. Every EU $d \in D$ observes the state information $s_d(t)$ during the time slot $t \in \mathbb{T}$. This state information consists of the task information, predicted position, current position, and predicted resource availability. Here, $s_d(t) \in \mathcal{S}$, where $\mathcal{S}$ represents the global agent state space. The observed state information $s_d(t)$ for EU d is given by the following expression:

$$s_d(t) = (Pos_{cur}(t), Pos_{pre}(t+1), task_d(t), Res(t+1)) \tag{20}$$

$Pos_{cur}(t)$ and $Pos_{pre}(t+1)$ respectively represent the current and next time slot geographical positions of the EU, while $task_d(t)$ represents task information, including task size, execution positions of preceding tasks and DAG information. At the same time, according to the DAG structure, its pre-task offloading results are obtained and used as part of the

[1]https://github.com/alibaba/clusterdata/tree/v2018/cluster-trace-v2018

observable state information. The alternative edge nodes set $\mathbb{P}$, where each EN's resource availability is represented as $Res_n(t+1) = \{CPU_n, Mem_n\}(n \in N)$.

Action. Due to the decomposition of tasks generated by edge users into several sub-tasks, a mapping between states and actions is established based on the description of the observed state for each sub-task. In other words, an offloading decision is made for each sub-task. EU d enters its locally observed state into Action-Net, and selects an action based on the output of Action-Net as follows:

$$a_d = \begin{cases} ActionNet(s_d(t)) + OUNoise & \text{if } e \leq \epsilon \\ ActionNet(s_d(t)) & \text{otherwise} \end{cases} \tag{21}$$

We discretized the decision about offloading into a two-dimensional action, represented as $ActionNet(s_d(t)) = \{\rho_d, \theta_d\}$. Here, ϵ represents the exploration rate, aiming to assist Actor-Net in exploring the optimal offloading strategy. When a random number $e \leq \epsilon$, OUNoise is added to the generated action to explore new actions. To ensure the stability of the algorithm, we continuously decay ϵ emphasizing more exploration in the early stages and reducing it later.

Reward. Edge users take actions and interact with the environment, generating rewards $Reward_d = r\{s_d(t), a_d(t)\}$. The global reward is represented as $\mathbb{R}$, calculated as follows:

$$\mathbb{R} = \sum_{d}^{D} r\{s_d(t), a_d(t)\} \tag{22}$$

$r\{s_d(t), a_d(t)\} = \delta * (Cost_d^{T_i} - Cost_n^{T_i})$, where $\delta = -1$ or 1 indicating the value of cost savings if the task is executed locally or at the ENs, respectively. At the beginning of the next time slot (i.e., time slot $t+1$), EU d observes the next state $s_d(t+1)$. Each EU stores its experience $(s_d(t), a_d(t), r_d(t), s_d(t+1))$ to the memory for training.

When the experience pool contains a sufficient number of samples, randomly sample a set of experiences from the memory. $\mathbb{S}$ represents the sample set, and $|\mathbb{S}|$ denotes the number of samples. Q_d and Q_d^{target} denote the Q_{value} obtained by critic network and target critic network, respectively. Update the critic network by minimizing the TD-Loss:

$$\mathcal{L}(\theta_d) = \frac{1}{|\mathbb{S}|} \sum (Q_d(s_d(t), a_d(t), \theta_d) - Q_{(d,t)}^{target})^2 \tag{23}$$

The minimization of the loss function is accomplished by performing backpropagation.

V. Experimental Evaluation

We design different sets of experiments to validate the effectiveness of the proposed method. The experiments aim to investigate the following four issues:

- RQ1: Is the user location prediction and available resource perception accurate?
- RQ2: Is the LSTM-MADDPG algorithm more effective in task-offloading decision-making compared to other reinforcement learning algorithms?
- RQ3: How does LSTM-MADDPG perform with different numbers of users?
- RQ4: How does the performance of the LSTM-MADDPG algorithm vary with different amounts of task data?

A. Experimental Setup

The PyTorch 1.9.1 framework is used to implement the proposed method. The model is trained with a computer with NVIDIA GTX1650Ti GPU, AMD Ryzen 7 CPU@2.90 GHz.The algorithm parameters are configured as shown in Table II. All experiments were performed with the same parameters, which are considered optimal settings based on our experimental observations.

TABLE II
ALGORITHM PARAMETERS

LSTM Parameters	Value	MADDPG Parameters	Value
Batch-size	64	Learning rate of actor	0.001
Number of layer	2	Learning rate of critic	0.001
Hidden-size	128	Experience replay buffer size	$1 * 10^5$
Learning rate	0.001	Mini-batch size	32
optimizer	Adam	Discount factor	0.9
Dropout	0.3	Update factor	0.01

B. Experiment Data

This experiment involves three data sets in the experiment.

- Data Set 1 is the Shanghai Telecom data set[2]. This dataset includes geographical location information for 3,233 base stations and user requests to the base stations.
- Data Set 2 is Shanghai Taxi Dataset[3]. This dataset contains the daily trajectory data for 4,316 taxis in Shanghai on February 20, 2007.
- Data Set 3 is Cluster-trace-v2018 Alibaba Cluster Dataset. This dataset covers the resource usage of 4,000 machines within eight days, describing the resource utilization of each machine and the DAG information for each job. This data set contains 4,201,007 jobs, in which the numbers of sub-tasks in each job are respectively 5, 6, 7, 9, 11, etc.
- Simulated Task data: Similar to existing studies [7], [8], we use simulation to generate task input/output as well as instruction sizes.

We selected 60 edge server locations from Dataset 1, as depicted in Fig.5. To represent the resource use of edge nodes, we mapped the machine resource usage information from Dataset 3. We consider each machine as an edge node, and the containers deployed on the machine as servers, with an average of 17 servers per edge node. The historical resource usage records of machines represent the resource utilization history of edge nodes. Each job is divided into several sub-tasks, and each sub-task is mapped to a machine based on its task ID. The taxi movement trajectories from Dataset 2 were used to model the mobility trajectories of edge users. The simulation parameters for the environment and tasks are provided below in Table III.

[2]http://sguangwang.com/TelecomDataset.html
[3]https://cse.hkust.edu.hk/scrg/

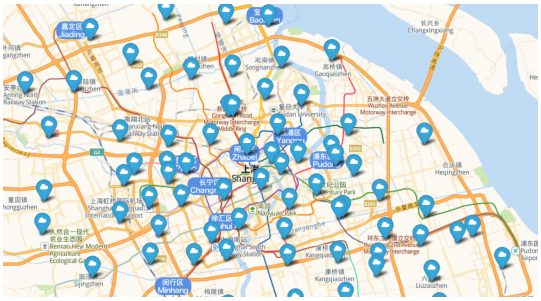

Fig. 5. Edge node distribution.

TABLE III
SIMULATION EXPERIMENT PARAMETER

Parameters	Value	Parameters	Value
fr_n	[31.8,51.8] GHz	fr_d	[31.8,51.8] GHz
mem_n	[32,128]GB	mem_d	[2,16]GB
$core_n$	[2,16]	$core_d$	1
$\mathcal{W}$	[25,35]Mbps	EC_d	[4,6] J/s
$\mathcal{H}_{d,n}$	[8.5,14]dbi	ET_d	[3,5] J/s
σ	-20dBm	$\mathcal{P}$	[13,33]dBm
$\mathcal{C}$	700 cycles/bit	I	[1,10]MB
Number of EN	60	O	[1,10]MB
		E	[50,100]MB

C. Experimental results

Predictive model accuracy. To address RQ1, we design an experiment to validate the accuracy of edge user trajectories and available resource perception. Firstly, we verified the accuracy of the EU's trajectory predictions by contrasting predicted positions with their actual locations. There were 1290 users in the test set. Table IV displays the user path prediction results. With an absolute error for all EU below 0.1, within an acceptable range, it indicates that LSTM can effectively predict user trajectories, ensuring the accuracy of trajectory predictions. To validate the usability of the trajectory prediction model, we assume that the vehicle's travel speed is 70 km/h. As shown in Fig.6, both the training and prediction time of the trajectory prediction model are less than the time it takes for the vehicle to move to the target position. Therefore, this trajectory prediction model can effectively support task offloading decisions.

We then confirmed that the resource availability prediction for edge nodes was accurate. We extracted data from 1000 machines from Dataset 3 for training and testing. Table V displays the forecasts of resource use for edge nodes. Prediction errors are within an acceptable range since the absolute error for every edge node is less than 0.1. The accuracy of resource forecasts has been guaranteed by using the resource-aware model to anticipate edge node resources.

Algorithm effectiveness. To address RQ2 and validate the effectiveness of the algorithm, we performed 1000 iteration episodes and evaluated the average reward values for several algorithms, including LSTM-MADDPG, MADDPG [28], DDQN [29], and DDPG [30]. Due to the need to construct precise mathematical analytical models in traditional methods, they are not suitable for dynamic scenarios where task offloading occurs during the mobility of edge users. Therefore, we compared various reinforcement learning algorithms. There were 10 EUs in the MEC environment. As shown in Fig.7, with each episode iteration, the average episode incentives for the LSTM-MADDPG, MADDPG, and DDQN offloading algorithms grew until they ultimately reached convergence reward values. Following a period of fluctuating reward amounts, we saw a notable rise, indicating that the networks continuously explored more optimal offloading strategies. In contrast, the DDQN algorithm stabilized its reward values after around 150 iterations. It is noteworthy that the dynamic instability of the multi-agent environment caused the DDPG algorithm, which is meant for single-agent situations, to become unsuccessful when applied to it. The reward values exhibited a declining trend before stabilizing. Overall, the method we proposed consistently achieved higher average rewards than the other algorithms, validating the effectiveness of our proposed algorithm.

TABLE IV
COMPARISON OF REAL AND PROJECTED VALUES FOR EU COORDINATES

Edge User	Real Coordinate	Predicate Coordinate	Absolute Error
U1	(121.4751, 31.2281)	(121.4764, 31.2272)	(0.013, 0.009)
U2	(121.4968, 31.3856)	(121.4949, 31.3841)	(0.019, 0.015)
U3	(121.6851, 31.2200)	(121.7211, 31.1930)	(0.036, 0.027)
All User	-	-	(0.037, 0.028)

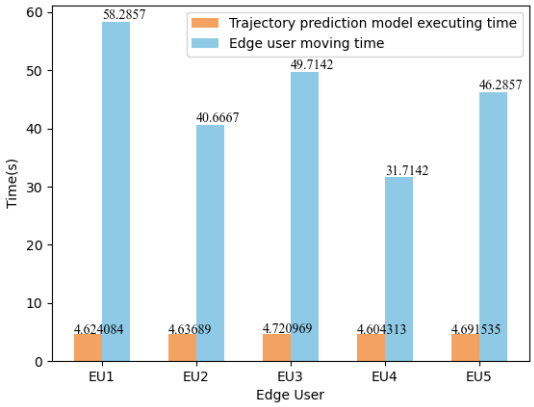

Fig. 6. Time consumption comparison between trajectory prediction model and EU movement.

TABLE V
COMPARISON OF ACTUAL AND PREDICTED VALUE OF EN RESOURCE USAGE

Edge Node	Real Resource(%)	Predicate Resource(%)	Absolute Error
N1	(29.00, 89.00)	(28.9651, 89.0892)	(0.0349, 0.0892)
N2	(30.00, 81.00)	(30.0854, 80.9159)	(0.0854, 0.0841)
N3	(24,00, 87.00)	(24.0181, 87.1069)	(0.0181, 0.1069)
All Nodes	-	-	(0.0693, 0.0846)

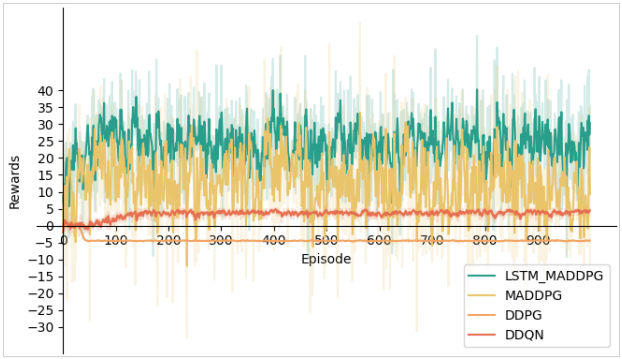

Fig. 7. Comparison of reward values for different algorithms.

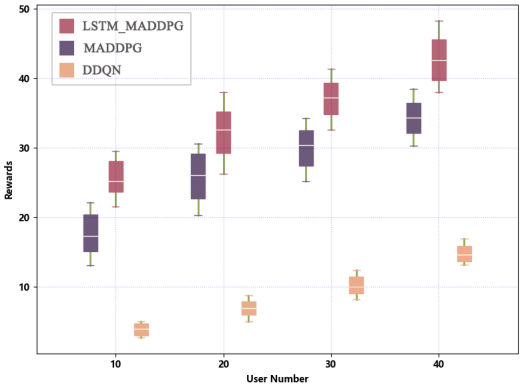

Fig. 8. Reward values under different user quantities.

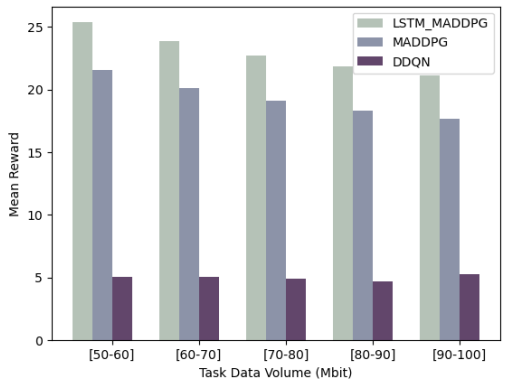

Fig. 9. Algorithm performance under different task data volumes.

Algorithm adaptability. To address RQ3 and examine the influence of the number of users on the algorithm's scalability, we designed a series of experiments to assess the algorithm's performance under various user quantities. We calculated the average episode payouts for 10, 20, 30, and 40 EUs, choosing 100 episodes after the algorithm's reward had converged. As illustrated in Fig.8, the global reward values increased as the number of users increased. This increase is attributed to the positive reward values generated by each user's action in response to the algorithm's decisions. Therefore, the worldwide reward values rose together with the number of users. It is noteworthy to note that as the number of users grew, the algorithm suggested in this research continually outperformed the baseline methods.

Algorithm performance. To address RQ4, we designed experiments to evaluate the algorithm's performance under different task data volumes. Fig.9, 10, and 11 illustrate the trends in average rewards, average task delay, and unit system energy consumption with increasing task data volumes. In this experiment, we controlled the number of users at 10, and task instruction sizes were set as [50,60] Mbit, [60,70] Mbit, [70,80] Mbit, [80,90] Mbit, and [90,100] Mbit. Fig.9 illustrates that the average rewards exhibit a diminishing trend with increasing task data volume. It is evident that when task data volume increases, task completion delay and energy usage increase as well, resulting in lower average rewards. LSTM-MADDPG regularly performs better than the other two baseline approaches. In Fig.10 and 11, LSTM-MADDPG, compared to MADDPG and DDQN, achieves savings of up to 14.03% and 39.50% in delay and 20.42% and 31.78% in energy consumption, respectively. Fig.10 indicates that as the task data volume increases, task completion delay continues to rise. This is attributed to the increased demand for computational resources for larger task volumes, coupled with the finite resources in the edge environment, leading to resource competition among users and an increase in task execution dealy. Similarly, as shown in Fig.11, the larger task volumes result in increased local computation energy consumption, and at the same time, more tasks are offloaded to edge nodes, leading to additional computational energy consumption.

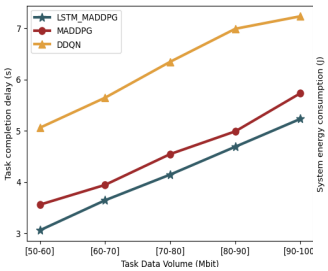

Fig. 10. Task execution delay.

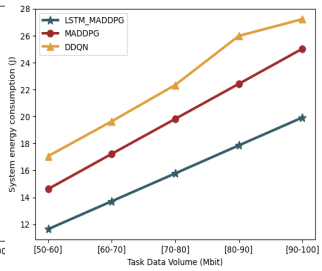

Fig. 11. Energy consumption per unit of task execution in the system.

VI. Conclusion

In a dynamic MEC environment, we have investigated the task offloading strategy with multi-user resource awareness and suggested the LSTM-MADDPG algorithm. This approach handles the task offloading difficulty in a multi-user competitive MEC environment with dynamically available resources(e.g. CPU and memory) by anticipating the paths of edge users and detecting the available resources of edge nodes. The approach is noteworthy because it addresses the intricacy of large tasks by breaking them down into smaller ones and mapping the connections between them into a DAG, which enables more precise task offloading. Numerous tests show that LSTM-MADDPG performs better than baseline techniques in terms of algorithmic rewards and system expenses. For future work, we will further explore these issues:1) We will conduct a more in-depth examination of task offloading in a hybrid MEC environment with competitive cooperation relationships. 2) We will further discuss methods to reduce the complexity of the proposed algorithm by incorporating deep compression [31] to decrease the number of multiplicative operations in the network structure.

ACKNOWLEDGMENTS

This work is funded by the National Natural Science Foundation of China under Grant No.62272145 and No.U21B2016, the Natural Science Research Startup Foundation of Recruiting Talents of Nanjing University of Posts and Telecommunications (Grant No. NY223166), and the Australian Research Council's Discovery Projects funding scheme (DP220101823).

REFERENCES

[1] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Gener. Comput. Syst.*, vol. 97, no. C, p. 219–235, aug 2019. [Online]. Available: https://doi.org/10.1016/j.future.2019.02.050

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[3] A. Islam, A. Debnath, M. Ghose, and S. Chakraborty, "A survey on task offloading in multi-access edge computing," *Journal of Systems Architecture*, vol. 118, p. 102225, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762121001570

[4] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A drl agent for jointly optimizing computation offloading and resource allocation in mec," *IEEE Internet of Things Journal*, vol. 8, no. 24, pp. 17 508–17 524, 2021.

[5] L. Ji and S. Guo, "Energy-efficient cooperative resource allocation in wireless powered mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4744–4754, 2019.

[6] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iraf: A deep reinforcement learning approach for collaborative mobile edge computing iot networks," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 7011–7024, 2019.

[7] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, vol. 21, no. 6, pp. 1985–1997, 2022.

[8] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 6, pp. 3425–3443, 2023.

[9] L. Zhao, E. Zhang, S. Wan, A. Hawbani, A. Y. Al-Dubai, G. Min, and A. Y. Zomaya, "Meson: A mobility-aware dependent task offloading scheme for urban vehicular edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–15, 2023.

[10] "Mobility and dependency-aware task offloading for intelligent assisted driving in vehicular edge computing networks," *Vehicular Communications*, vol. 45, p. 100720, 2024.

[11] J. Fang, D. Qu, H. Chen, and Y. Liu, "Dependency-aware dynamic task offloading based on deep reinforcement learning in mobile edge computing," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2023.

[12] L. Zhang, B. Cao, Y. Li, M. Peng, and G. Feng, "A multi-stage stochastic programming-based offloading policy for fog enabled iot-ehealth," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 411–425, 2021.

[13] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 6, pp. 4177–4190, 2018.

[14] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2019.

[15] L. Tang, B. Tang, L. Kang, and L. Zhang, "A novel task caching and migration strategy in multi-access edge computing based on the genetic algorithm," *Future Internet*, vol. 11, no. 8, 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/8/181

[16] H. Wu, Y. Sun, and K. Wolter, "Energy-efficient decision making for mobile cloud offloading," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 570–584, 2020.

[17] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 64–69, 2019.

[18] Z. Wu and D. Yan, "Deep reinforcement learning-based computation offloading for 5g vehicle-aware multi-access edge computing network," *China Communications*, vol. 18, no. 11, pp. 26–41, 2021.

[19] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, and K. Li, "Distributed task migration optimization in mec by extending multi-agent deep reinforcement learning approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1603–1614, 2021.

[20] F. Turčinović, G. Šišul, and M. Bosiljevac, "Lorawan base station improvement for better coverage and capacity," *Journal of Low Power Electronics and Applications*, vol. 12, no. 1, 2022. [Online]. Available: https://www.mdpi.com/2079-9268/12/1/1

[21] J. Liu, S. Guo, Q. Wang, C. Pan, and L. Yang, "Optimal multi-user offloading with resources allocation in mobile edge cloud computing," *Comput. Netw.*, vol. 221, no. C, feb 2023. [Online]. Available: https://doi.org/10.1016/j.comnet.2022.109522

[22] S. Xia, Z. Yao, Y. Li, and S. Mao, "Online distributed offloading and computing resource management with energy harvesting for heterogeneous mec-enabled iot," *IEEE Transactions on Wireless Communications*, vol. 20, no. 10, pp. 6743–6757, 2021.

[23] L. Chen, Y. Liu, Y. Lu, and H. Sun, "Energy-aware and mobility-driven computation offloading in mec," *J. Grid Comput.*, vol. 21, no. 2, apr 2023. [Online]. Available: https://doi.org/10.1007/s10723-023-09654-1

[24] L. Li, T. Q. Quek, J. Ren, H. H. Yang, Z. Chen, and Y. Zhang, "An incentive-aware job offloading control framework for multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 1, pp. 63–75, 2021.

[25] A. J. Goldsmith, "Wireless communications," 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:10928379

[26] Q.-V. Pham, L. B. Le, S.-H. Chung, and W.-J. Hwang, "Mobile edge computing with wireless backhaul: Joint task offloading and resource allocation," *IEEE Access*, vol. 7, pp. 16 444–16 459, 2019.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6382–6393.

[29] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 2094–2100.

[30] Y. Liang, Y. He, and X. Zhong, "Decentralized computation offloading and resource allocation in mec by deep reinforcement learning," in *2020 IEEE/CIC International Conference on Communications in China (ICCC)*, 2020, pp. 244–249.

[31] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *arXiv: Computer Vision and Pattern Recognition*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:2134321