Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.058810

ARTICLE

Dynamic Task Offloading Scheme for Edge Computing via Meta-Reinforcement Learning

Jiajia Liu[1,*], Peng Xie[2], Wei Li[2], Bo Tang[2] and Jianhua Liu[2]

[1]Teacher Development and Teaching Evaluation Center, Civil Aviation Flight University of China, Guanghan, 618307, China

[2]Institute of Electronics and Electrical Engineering, Civil Aviation Flight University of China, Guanghan, 618307, China

*Corresponding Author: Jiajia Liu. Email: cafucljj@cafuc.edu.cn

Received: 21 September 2024 Accepted: 22 November 2024 Published: 17 February 2025

ABSTRACT

As an important complement to cloud computing, edge computing can effectively reduce the workload of the backbone network. To reduce latency and energy consumption of edge computing, deep learning is used to learn the task offloading strategies by interacting with the entities. In actual application scenarios, users of edge computing are always changing dynamically. However, the existing task offloading strategies cannot be applied to such dynamic scenarios. To solve this problem, we propose a novel dynamic task offloading framework for distributed edge computing, leveraging the potential of meta-reinforcement learning (MRL). Our approach formulates a multi-objective optimization problem aimed at minimizing both delay and energy consumption. We model the task offloading strategy using a directed acyclic graph (DAG). Furthermore, we propose a distributed edge computing adaptive task offloading algorithm rooted in MRL. This algorithm integrates multiple Markov decision processes (MDP) with a sequence-to-sequence (seq2seq) network, enabling it to learn and adapt task offloading strategies responsively across diverse network environments. To achieve joint optimization of delay and energy consumption, we incorporate the non-dominated sorting genetic algorithm II (NSGA-II) into our framework. Simulation results demonstrate the superiority of our proposed solution, achieving a 21% reduction in time delay and a 19% decrease in energy consumption compared to alternative task offloading schemes. Moreover, our scheme exhibits remarkable adaptability, responding swiftly to changes in various network environments.

KEYWORDS

Edge computing; adaptive; meta; task offloading; joint optimization

1 Introduction

Cloud computing, as a centralized computing model, provides powerful computing and storage capabilities through remote data centers and is widely adopted [1,2]. Edge computing, as a paradigm of cloud computing, provides distributed computing services at the network's edge, addressing the growing need for real-time processing and large datasets in various application scenarios [3–6]. Mobile Edge Computing (MEC) deploys server at the network edge to handle intensive computing and data processing tasks, effectively reducing backhaul link load and minimizing service delays [7–9]. However, diverse terminal devices and task types bring challenges to task offloading decision. How to allocate tasks reasonably to achieve efficient utilization of computing resources has become a very urgent challenge.

The early research mainly focused on the static optimization of task offloading decision, and determined the task allocation strategy by offline methods. However, these methods are often unable to adapt to the dynamic network environment and task characteristics. Literature [10] proposes a scheme based on heuristic algorithm to construct offloading strategy, but the proposed scheme relies too much on the expert strategy of MEC system, adapting a specific heuristic algorithm to the dynamic MEC landscape, influenced by the growing complexity of MEC applications and architecture, can be challenging, and it is impossible to adapt the learning strategy to different network environments. Literature [11] introduced a strategy calculation offloading algorithm utilizing a double-depth Q-network (DKN). This algorithm learns the optimal strategy without prior knowledge of network dynamics, but the scheme could not adapt to the new network environment. Literature [12] introduced an offloading algorithm using deep meta reinforcement learning. This algorithm utilizes multiple parallel deep neural networks (DNN) and Q learning to make precise decisions, swiftly adapting to dynamic environments to obtain the optimal offloading strategy, but this scheme does not consider the analysis of each detailed index in edge calculation. Hence, a multi-objective optimization meta-reinforcement learning algorithm is needed to adapt to diverse network environments and simultaneously optimize time delay and energy consumption.

This paper suggests an adaptive task offloading scheme for edge computing to address dynamic environment task offloading challenges, utilizing the concept of meta-strategy. A novel approach is introduced, leveraging the MRL algorithm by combining it with a seq2seq neural network. This innovative method focuses on learning a meta-offload strategy applicable to all User Equipment (UE), enabling the rapid derivation of efficient strategies for individual UEs through the integration of meta-strategy and local data. To assess the dynamic task offloading meta-reinforcement learning (DTOML)'s performance in dynamic scenarios, diverse situations are considered, involving heterogeneous users with distinct mobile application preferences represented as DAGs varying in height, width, and task numbers. Additionally, the transmission rate is adjusted based on the proximity of UEs to MEC hosts. We explore how to use machine learning and decision optimization algorithms to adaptively adjust the task allocation strategy by learning historical data and monitoring network status in real time. We will design and implement a meta-strategy model, and learn and update the strategy by analyzing the task characteristics and network status, so as to provide the best task processing performance.

In view of these observations, we propose a meta-strategy-based dynamic task offloading scheme for distributed edge computing (DTOML), which establishes an offloading decision through DAG, then establishes different network environment models through multiple MDPs, and then learns and updates the offloading strategy through seq2seq network to adapt to the dynamic network environment and task requirements. Finally, the NSGA-II is employed to optimize task scheduling by jointly addressing delay and energy consumption. Our approach demonstrates reduced latency and energy consumption compared to alternative schemes.

In addition, this scheme has good application value. First, it can be used in the traffic management system. Through edge computing task offloading, traffic flow can be analyzed in real time, congestion can be predicted, and traffic signals can be dynamically adjusted to improve urban traffic efficiency. Secondly, it can be used in the intelligent driving system. Through edge computing task offloading, it can quickly process the perception data of the vehicle's surrounding environment, support the auto drive system to make decisions, and improve safety and reaction speed.

The main contributions are as follows:

(1) This paper proposes a dynamic task offloading scheme for edge computing based on MRL. By modeling the task offloading decision as a DAG, different edge computing network environments are modeled by using multiple markov decision processes (MDP), and the "outer loop" training of offloading strategy is carried out by using Proximal Policy Optimization (PPO) strategy gradient method in deep reinforcement learning.
(2) To further improve the self-adaptability of task offloading scheme, introduces seq2seq network to carry out self-adaptive "inner-loop" training of offloading strategy. By analyzing historical data and real-time network status, the seq2seq network can adaptively learn and update offloading strategies in response to dynamic changes in the network environment and task requirements.
(3) Aiming at two important indexes, time delay and energy consumption, this paper adopts multi-objective optimization algorithm for joint optimization. By weighing the relationship between the two, a set of balanced offloading strategies is found.

The overall structure is as follows: The Section 2 expounds the related work of this paper; In Section 3, the edge calculation model is put forward and the problems studied are expounded. In Section 4, an adaptive task offloading scheme for distributed edge computing based on meta-strategy is proposed. In Section 5, the performance is evaluated. Finally, we conclude our paper in Section 6.

2 Related Work

This section describes the related work on edge computing, meta-reinforcement learning and multi-objective optimization.

2.1 Edge Computation

In recent years, people have recognized edge computing as an emerging network paradigm. It alleviates the strain caused by the unprecedented surge in traffic and computing demand by offering cloud services at the network edge. The surge of mobile devices for emerging applications such as healthcare, media recognition, intelligent transportation systems (ITS), and drones is driving the rapid advancement and optimization of edge computing technology [13].

MEC plays a crucial role in task offloading, specifically in computing offloading. In order to find the best strategy, Al-Shuwaili et al. [14] proposed a scheme to model the related tasks that organize the mobile application as a DAG, but did not solve the key problem that the offloading of the DAG model with minimum delay in MEC system depends on tasks. To solve the resource allocation problem in MEC systems, Lyu et al. [15] proposed a resource allocation scheme of Stackelberg game model with multi-leaders and multi-followers. Edge nodes can set the price of edge computing resources according to other nodes' strategies and predict users' behaviors. Then, end users can choose their own optimal strategy according to the value strategy of edge nodes, but it still does not solve the problem that the offloading strategy cannot adapt to various network environments. To solve the problem that the offloading strategy cannot adapt to various network environments, Wang et al. [16] introduced a task offloading approach using meta-reinforcement learning, enabling rapid adaptation to new environments with minimal gradient updates and samples, and the learning agent interacts with different network environments to train an offloading strategy that can quickly adapt to new environments, unfortunately, this method only optimizes time delay. To solve the problem of geographical dispersion of edge computing, Hu et al. [17] proposed an adaptive scheduling method to deal with dynamic real-time computing requests, reducing the complexity of decentralization work to minimize the energy consumption, while this method only optimizes the energy consumption.

Additionally, Dong et al. [18] proposed an algorithm based on quantum particle swarm optimization (QPSO) to minimize energy consumption. Liu et al. [19] introduced a multi-master multi-slave bi-level game model based on task offloading and pricing strategies for edge computing in vehicular networks. Jeremiah et al. [20] achieved edge node collaboration by integrating digital twin technology (DT).

2.2 Meta-Reinforcement Learning

DRL integrates reinforcement learning (RL) with DNNs, offering a promising solution to the mentioned challenges. DRL excels at addressing intricate problems through trial-and-error learning, even in the absence of a precise environmental model [21–23]. In order to realize an efficient task offloading, Zhu et al. [24] introduced a scheme to formalize the offloading problem, framing it as an optimization challenge involving energy and time based on users' experience. The strategy is derived using deep Q learning (DQN), but it did not consider the adaptability of the strategy to different network environments. The aforementioned solutions lack adaptability to unforeseen disturbances or unknown scenarios (e.g., a new environment). Their low sample efficiency and the requirement for extensive retraining hinder their ability to cope with changes in applications, tasks, or data rates. Consequently, they spend a considerable amount of time learning new strategies for the updated environment.

Meta-reinforcement learning accelerates the learning of new tasks by leveraging past experiences in a sequence of learning tasks [25,26], and strives to acquire strategies by leveraging prior experience, effectively tackling new tasks with minimal interaction in the environment. The application of MRL to address the computational offloading issue yields evident advantages. We can rapidly acquire tailored strategies for novel mobile users based on local data and meta-strategies. To address the challenge of the offloading strategy's lack of adaptability to the evolving network environment, Botvinick et al. [27] introduced the "inner loop" and "outer loop" training approach for acquiring the optimal offloading strategy. The "outer loop" leverages its experience across various task contexts to iteratively fine-tune the parameters of the meta-strategy controlling the "inner loop." Utilizing the meta-strategy, the "inner loop" can rapidly adjust to new tasks with minimal gradient updates. However, this approach does not explicitly consider its applicability in the edge computing environment. To solve the problem of how to choose the most suitable neural network architecture among many types of deep learning architectures and how to optimize the superparameter of the selected neural network, Thar et al. [28] proposed a deep learning model deployment scheme based on meta-reinforcement learning to build a suitable model for self-prediction of content popularity, and proposed a feedback mechanism to find a prediction model, but it did not compare its performance with the latest meta-reinforcement learning algorithm scheme. Merging deep learning for perception with reinforcement learning for decision-making, and meta-learning's rapid environmental adaptation, an offloading strategy can be efficiently derived in dynamic environments. However, this approach solely focuses on offloading efficiency without delving into a comprehensive analysis of various detailed metrics in edge computing. To solve the problem of computing task scheduling in heterogeneous cellular systems, Niu et al. [29] proposed a multi-agent element PPO algorithm extended to non-stationary heterogeneous edge computing systems, so as to quickly adapt the control strategy learning to non-stationary, but this method does not involve the joint optimization of multiple task indicators. To minimize processing delays and energy consumption in a collaborative edge computing system with multiple users and sites, Avgeris et al. [30] proposed a two-stage mechanism based on reinforcement learning. In the first stage, mobile devices (MDs) autonomously decide whether to offload tasks to edge sites (ES) through an iterative reinforcement learning mechanism, thereby reducing energy consumption for compute-intensive tasks. In the second stage, if some ES are overloaded, task collaboration offloading between sites is implemented using DQN, transferring tasks to less loaded sites. However, this method also does not consider the joint optimization of multiple task metrics and relies on an offline-trained DQN model, which may lack adaptability in dynamic network environments. To address the impact of dynamic changes in the MEC on the generalization ability of task offloading algorithms, Li et al. [31] proposed a MRL algorithm, can adapting faster to new environments, And dynamically adjust the priority of the sampling task during the training process to improve the sampling efficiency of the task, but this algorithm also does not involve how to balance and optimize multiple task indicators for task offloading. Hence, this paper presents a dynamic method with meta-reinforcement learning. The method integrates a directed acyclic graph, multiple Markov strategies, adaptive training network, and a multi-objective optimization algorithm to address the mentioned challenges.

In summary, while the current MRL-based offloading strategies demonstrate many advantages, they still have some limitations. Firstly, many existing MRL methods focus on a single performance metric, lacking a comprehensive consideration of multi-objective optimization, which restricts their applicability in complex environments. Secondly, many MRL methods rely on prior experience for policy learning, but in dynamically changing environments, the initial experience may not sufficiently cover all situations, leading to suboptimal performance in new tasks or environments. Additionally, some MRL algorithms exhibit poor adaptability when faced with complex tasks, and their update frequency and adaptability may not meet the demands of rapidly changing network environments.

2.3 Multi-Objective Optimization

Multi-objective optimization algorithm is to make multiple objectives to be optimized as best as possible at the same time under given constraints, and the solution is usually a set of equilibrium solutions [32–36]. To enhance the estimation accuracy of range vector jump technology, Verma et al. [36] suggested using NSGA-II algorithm to solve the selected multi-class combinatorial optimization problem, and elaborated the multi-class combinatorial optimization problem in detail. To reduce the positioning error, Wang et al. [37] proposed a localization algorithm based on NSGA-II, however, the experimental part of this scheme lacks effectiveness verification for dynamically changing topological structures. To solve the joint optimization of reservoir operation problem and benchmark problem, Liu et al. [38] proposed a new bi-objective algorithm based on NSGA-II algorithm, and then tested its performance in the bi-objective optimization through benchmark problem and reservoir operation problem, but did not apply it to practical problems, such as edge computing. To solve the task offloading problem under 6G networks, Long et al. [39] proposed an algorithm based on the "end-to-end cloud" architecture. However, this algorithm does not consider completely dispersed network environments, But it is validated in a collaborative network environment with edge clouds. To solve the workflow offloading problem in multi-objective optimization, Pan et al. [40] proposed a clustering evolutionary algorithm to minimize the cost and energy consumption. However, the optimization effect of this algorithm is not significantly improved.

Based on the above research, this paper proposes a distributed edge computing adaptive task offloading solution (DTOML) based on a meta-strategy. This solution establishes offloading decisions through DAG, builds different network environment models using multiple MDPs, and subsequently employs a seq2seq network to learn and update offloading strategies, adapting to the dynamically changing network environment and task demands. Seq2seq networks have been widely applied in previous works, including in natural language processing [41] and time series prediction [42], particularly excelling in problems that require handling continuous data or multi-step decision-making. In reinforcement learning, seq2seq helps RL agents generate suitable action sequences [43], effectively addressing multi-step task planning and the decomposition of complex tasks. Finally, the NSGA-II algorithm is used to jointly optimize the delay and energy consumption during the task scheduling process. The NSGA-II algorithm has demonstrated outstanding performance in previous multi-objective optimization problems, particularly in scenarios such as communication networks, resource allocation, and multi-objective task scheduling [44]. It effectively balances multiple conflicting optimization objectives and shows high efficiency and stability in solving task offloading problems. Compared with other solutions, this approach achieves lower latency and reduced energy consumption.

3 System Model and Overview of the Problem

3.1 System Overview

As shown in Fig. 1, we consider two scenarios: an increase in the number of users and a decrease in the number of users. We consider three network environments, namely, environments A, B and C, and each environment is composed of multiple user equipments. When increasing, environment A becomes environment B; and, when decreasing, environment B becomes environment C. For network environment A, its composition is expressed as: $N_{A-total} = \{1, 2, \ldots, N_A\}$. In time slot t, the device generates a set of tasks, which are expressed as: $A_i^t = \{a_{i,1}, \ldots, a_{i,K_A(i,t)}\}$, $K_A(i, t)$ is the total number of tasks of device i in network environment A, the each device's task generation process can be represented by a Poisson process. The size of task $a_{i,k}$ is denoted by $s_{i,k}$, the CPU cycle per second required for computing task $a_{i,k}$ is denoted by $c_{i,k}$. For task $a_{i,k}$, device i can offload it to device j or process it locally. The addition of new devices will cause changes in the network structure, and network environment A will become network environment B.

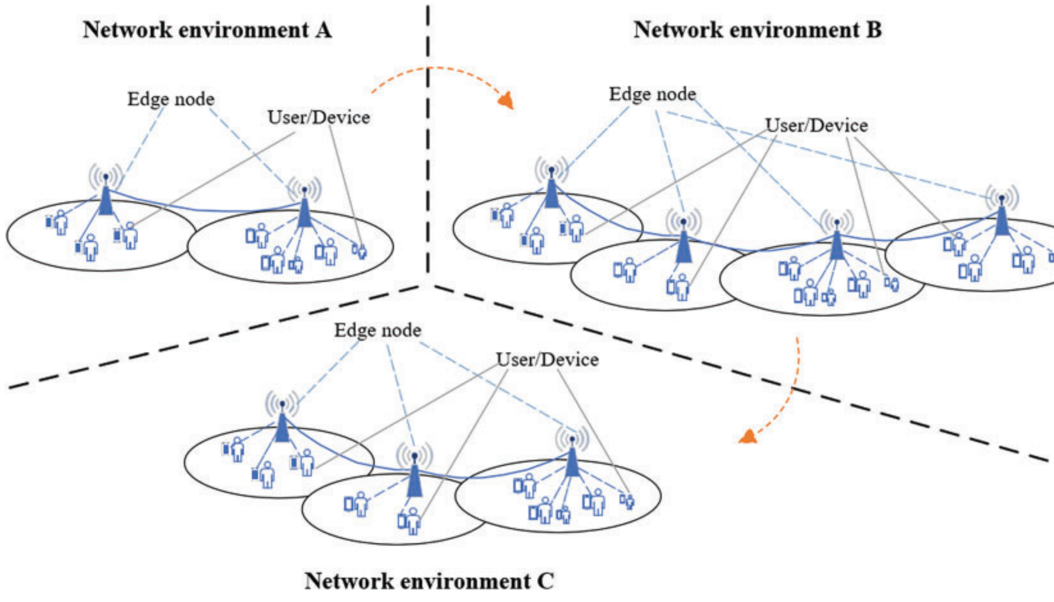


Figure 1: System model diagram

For network environment B, its composition is expressed by $N_{B-total} = \{1, 2, \ldots, N_B\}$. The device generates a set of tasks, which are expressed as: $B_i^t = \{b_{i,1}, \ldots, b_{i,K_B(i,t)}\}$, $K_B(i, t)$ is the total number of tasks of device i in network environment B. In network environment B, the departure of existing edge devices will lead to network environment B being transformed into another new network environment C. For network environment C, its composition is expressed as: $N_{C-total} = \{1, 2, \ldots, N_C\}$. The device generates a set of tasks, which are expressed as: $C_i^t = \{c_{i,1}, \ldots, c_{i,K_C(i,t)}\}$, here $K_C(i, t)$ is the total number of tasks of device i in network environment C. A dynamic network environment is built through the entry and exit of devices, constructing three different network environments A, B, and C with different numbers of devices. The main symbol descriptions involved in this paper are shown in Table 1.

Table 1: The symbol description involved in this paper

Notation	Description
$T_{i,k}^{UE}$	Local execution delay of task $a_{i,k}$
$E_{i,k}^{UE}$	Local execution energy consumption of task $a_{i,k}$
$T_{i,k}^{up}$	The UE sends the task $a_{i,k}$ delay to the MEC host through the wireless transmission channel
$T_{i,k}^{c}$	Execution delay of task $a_{i,k}$ on MEC
$T_{i,k}^{down}$	Receiving delay of task $a_{i,k}$ on UE
$E_{i,k}^{UL}$	Offloading energy consumption of task $a_{i,k}$
$FT_{i,k}^{UE}$	Completion time of task $a_{i,k}$ on UE under DAG model
$FT_{i,k}^{up}$	Transmission time of task $a_{i,k}$ on wireless uplink channel under DAG model
$FT_{i,k}^{MEC}$	Completion time of task $a_{i,k}$ on MEC DAG model
$FT_{i,k}^{down}$	Transmission time of task $a_{i,k}$ on wireless downlink channel under DAG model
$\Upsilon_{i,k}^{UE}$	Resource availability time of task $a_{i,k}$ on UE
$\Upsilon_{i,k}^{up}$	Resource availability time of task $a_{i,k}$ on wireless uplink channel
$\Upsilon_{i,k}^{MEC}$	Resource availability time of task $a_{i,k}$ on MEC
$\Upsilon_{i,k}^{down}$	Resource availability time of task $a_{i,k}$ on wireless downlink channel
$\pi_{\theta_{i,k}}$	Target strategy of learning task $a_{i,k}$
$\pi_{\theta_{i,k}^{o}}$	Sampling strategy for generating the initial trajectory of learning task $a_{i,k}$

3.2 Communication and Computing Model

In distributed computing offloading, end-to-end delay includes local processing delay, uplink transmission delay, downlink transmission delay and remote processing delay.

If the task $a_{i,k}$ is executed locally, the delay of processing the task $a_{i,k}$ is only the local execution delay $T_{i,k}^{UE}$, which is calculated by the following formula:

$$T_{i,k}^{UE} = \frac{C_{i,k}}{f_{UE}}, \tag{1}$$

where $C_{i,k}$ is the CPU cycle required for computing task $a_{i,k}$, f_{UE} is the computing power of the UE. $E_{i,k}^{UE}$ denoted the energy consumption when task $a_{i,k}$ is processed and calculated locally, which is calculated by the following formula:

$$E_{i,k}^{UE} = f_{UE} T_{i,k}^{UE}. \tag{2}$$

If the task $a_{i,k}$ is offloaded, the UE need to send the task $a_{i,k}$ to the MEC host through wireless transmission channel, and the transformation delay $T_{i,k}^{up}$ of the task $a_{i,k}$ is calculated by the following formula:

$$T_{i,k}^{up} = \frac{s_{i,k}^{send}}{R^{up}}, \tag{3}$$

where $s_{i,k}^{send}$ is the size of k-th subtask of $a_{i,k}$. The transmission rate of wireless uplink channel is denoted as R^{up}. Then, the MEC terminal executes the receiving party of $a_{i,k}$, and the delay $T_{i,k}^{c}$ executing the task $a_{i,k}$ at the MEC terminal is calculated by the following formula:

$$T_{i,k}^{c} = \frac{C_{i,k}}{f_{MEC}}, \tag{4}$$

where f_{MEC} is the computing power of the MEC device. Finally, the execution result of the task $a_{i,k}$ is returned to the UE, and the reception delay $T_{i,k}^{down}$ of the task $a_{i,k}$ is calculated by the following formula:

$$T_{i,k}^{down} = \frac{s_{i,k}^{r}}{R^{down}}, \tag{5}$$

where $s_{i,k}^{r}$ denotes the size of the result of processing $a_{i,k}$, the transmission rate of the downlink is denoted as R^{down}.

The energy consumption $E_{i,k}^{UL}$ offloading the task $a_{i,k}$ includes two parts:

(1) $E_{i,k}^{tran}$ of the task $a_{i,k}$ in the transmission process, including the transmission energy consumption $E_{i,k}^{up}$ of the task $a_{i,k}$ in the uplink and the reception energy consumption $E_{i,k}^{down}$ in the downlink, namely: $E_{i,k}^{tran} = E_{i,k}^{up} + E_{i,k}^{down}$.

(2) The calculation energy consumption $E_{i,k}^{MEC}$ at the edge devices.

Therefore, the energy consumption $E_{i,k}^{UL}$ offloading the task $a_{i,k}$ can be calculated by the following formula:

$$\begin{aligned} E_{i,k}^{UL} &= E_{i,k}^{tran} + E_{i,k}^{MEC} \\ &= (E_{i,k}^{up} + E_{i,k}^{down}) + E_{i,k}^{MEC} \\ &= (R^{up} T_{i,k}^{up} + R^{down} T_{i,k}^{down}) + f_{MEC} T_{i,k}^{c}. \end{aligned} \tag{6}$$

3.3 Problem Description

We consider that the task offloading decision is a DAG offloading plan, that is, $G = (A, D)$, where the set of offloading tasks is denoted as A and the dependency between tasks is denoted by D. For example, $\vec{d} = (a_{i,k}, a_{j,k})$ means that task $a_{i,k}$ is the parent task of task $a_{j,k}$. Subject to dependencies, subtasks must wait until all their parent tasks are finished.

If the task $a_{i,k}$ is processed locally at the UE side, the start execution time of the task $a_{i,k}$ depends on the completion time of its parent task $a_{j,k}$ and the available time of the UE. The completion time $FT_{i,k}^{UE}$ of task $a_{i,k}$ on the UE side is calculated by:

$$FT_{i,k}^{UE} = \max\{\Upsilon_{i,k}^{UE}, \max_{j \in parent}\{FT_{j,k}^{UE}, T_{i,k}^{down}\}\} + T_{i,k}^{UE}, \tag{7}$$

$$\Upsilon_{i,k}^{UE} = \max\{\Upsilon_{i-1,k}^{UE}, FT_{i-1,k}^{UE}\}. \tag{8}$$

where the resource available time of task $a_{i,k}$ on the UE is denoted as $\Upsilon_{i,k}^{UE}$, the completion time of the parent $a_{j,k}$ of $a_{i,k}$ on the UE side, the completion time of parent task $a_{j,k}$ on downlink is denoted as $T_{i,k}^{down}$, the delay of task $a_{i,k}$ on UE side is denoted by $T_{i,k}^{UE}$, the resource available time of task $a_{i-1,k}$ on the UE is denoted as $\Upsilon_{i-1,k}^{UE}$, the completion time of task $a_{i-1,k}$ on UE is denoted as $FT_{i-1,k}^{UE}$.

Therefore, the energy consumption $FE_{i,k}^{UE}$ of task $a_{i,k}$ in local processing is calculated by the following formula:

$$FE_{i,k}^{UE} = f_{UE} FT_{i,k}^{UE}. \tag{9}$$

If task $a_{i,k}$ is offloaded to the MEC host, then $a_{i,k}$ can only initiate data transmission after completing its parent tasks and ensuring the availability of the uplink channel. Therefore, the completion time $FT_{i,k}^{up}$ on the wireless uplink can be calculated by the following formula:

$$FT_{i,k}^{up} = \max\{\Upsilon_{i,k}^{up}, \max_{j \in parent}\{FT_{j,k}^{UE}, T_{i,k}^{down}\}\} + T_{i,k}^{up}, \tag{10}$$

$$\Upsilon_{i,k}^{up} = \max\{\Upsilon_{i-1,k}^{up}, FT_{i-1,k}^{up}\}. \tag{11}$$

where the resource availability time of task $a_{i,k}$ on uplink channel is denoted as $\Upsilon_{i,k}^{up}$, the delay of task $\Upsilon_{i,k}^{up}$ on uplink channel is denoted as $T_{i,k}^{up}$, the resource availability time of task $a_{i-1,k}$ on uplink channel is denoted as $\Upsilon_{i-1,k}^{up}$, the transmission time of task $a_{i-1,k}$ on uplink channel is denoted by $FT_{i,k}^{up}$.

The completion time $FT_{i,k}^{MEC}$ of task $a_{i,k}$ on MEC terminal is calculated by the following formula:

$$FT_{i,k}^{MEC} = \max\{\Upsilon_{i,k}^{MEC}, \max_{j \in parent}\{FT_{j,k}^{up}, T_{i,k}^{MEC}\}\} + T_{i,k}^{MEC}, \tag{12}$$

$$\Upsilon_{i,k}^{MEC} = \max\{\Upsilon_{i-1,k}^{MEC}, FT_{i-1,k}^{MEC}\}. \tag{13}$$

where $\Upsilon_{i,k}^{MEC}$ denotes the resource availability time of task $a_{i,k}$ on MEC side, the transmission time of task $a_{i,k}$ on uplink channel is denoted as $FT_{i,k}^{up}$, the transmission time of parent task $a_{j,k}$ of task $a_{i,k}$ on MEC side is denoted as $FT_{j,k}^{up}$, the delay of task $a_{i,k}$ at MEC end is denoted as $\Upsilon_{i-1,k}^{MEC}$, the resource availability time of task $a_{i-1,k}$ on MEC side is denoted as $\Upsilon_{i-1,k}^{MEC}$, the completion time of task $a_{i-1,k}$ on MEC side is denoted as $FT_{i-1,k}^{MEC}$, and the completion time of task $a_{i,k}$ on MEC side is denoted by $FT_{i,k}^{MEC}$.

The transmission time $FT_{j,k}^{down}$ of task $a_{i,k}$ on the wireless downlink channel is calculated by the following formula:

$$FT_{i,k}^{down} = \max\{\Upsilon_{i,k}^{down}, FT_{i,k}^{MEC}\} + T_{i,k}^{down}, \tag{14}$$

$$\Upsilon_{i,k}^{down} = \max\{\Upsilon_{i-1,k}^{down}, FT_{i-1,k}^{down}\}. \tag{15}$$

where the resource availability time of task $a_{i,k}$ on downlink channel is denoted by $FT_{i,k}^{down}$, the delay of task $a_{i,k}$ on the downlink is denoted by $T_{i,k}^{down}$.

The total energy consumption $FE_{i,k}^{UL}$ when task $a_{i,k}$ is offloaded is calculated by the following formula:

$$\begin{aligned} FE_{i,k}^{UL} &= FE_{i,k}^{tran} + FE_{i,k}^{MEC} \\ &= (FE_{i,k}^{up} + FE_{i,k}^{down}) + FE_{i,k}^{MEC} \\ &= (R^{up} FT_{i,k}^{up} + R^{down} FT_{i,k}^{down}) + f_{MEC} FT_{i,k}^{c}. \end{aligned} \tag{16}$$

The total delay and energy consumption are calculated using the following formulas, respectively:

$$FT_{i,k} = \max_{a_{i,k} \in \kappa}\{FT_{i,k}^{UE}, FT_{i,k}^{down}\}, \tag{17}$$

$$FE_{i,k} = \max_{a_{i,k} \in \kappa}\{FE_{i,k}^{UE}, FE_{i,k}^{down}\}. \tag{18}$$

where the exit the collection of tasks is denoted as κ, that is, the set of task $a_{i,k}$ that has been completed.

The ultimate goal of this article is to find a fast and effective offloading plan for DAG, which minimizes the total time delay and total energy consumption of the entire system. Therefore, we describe the task offloading process as a multi-objective optimization problem with two objective functions. The total delay $FT_{i,k}$ of Eq. (17) and energy consumption $FE_{i,k}$ of Eq. (18) are reduced to the equilibrium minimum by adding a weight factor α, so as to obtain the total target value Q, that is: the minimization of the total delay denoted as $FT_{i,k}$ in Eq. (17) and the total energy consumption denoted as $FE_{i,k}$ in Eq. (18) is achieved by introducing a weight factor α. This optimization process aims to balance both metrics to reach an equilibrium state, ultimately resulting in the minimization of the overall objective function denoted as Q. Mathematically, this can be expressed as:

$$\text{P1}: \quad Q = \min\{\alpha FT_{i,k} + (1-\alpha)FE_{i,k}\}. \tag{19}$$

$$\text{s.t.} \quad f_{UE} \le f_{UE-max}, \tag{20}$$

$$f_{MEC} \le f_{MEC-max}, \tag{21}$$

$$FT_{i,k}^{up} + FT_{i,k}^{MEC} + FT_{i,k}^{down} \le T_{max}, \tag{22}$$

$$FE_{i,k}^{tran} + FE_{i,k}^{MEC} \le E_{max}. \tag{23}$$

Due to the latency and energy consumption in problem P1, which are closely related to drones and users, and will increase with the increase of users and drones, the action and state space of the offloading strategy will grow exponentially, making it an NP hard problem that cannot be solved by traditional methods. Therefore, this paper proposes the DTOML algorithm to solve this problem.

4 Meta-Policy-Based Distributed Edge Computing Adaptive Task Offloading Scheme (DTOML)

In this section, a scheme based on meta-strategy (DTOML) is introduced. Firstly, an overview of the scheme is provided, and then the scheme is described in detail.

4.1 Overview of Algorithm

The DTOML algorithm is divided into two training cycles: the "inner loop" training of task-specific strategy at UE end and the "outer loop" training of meta-strategy at MEC end.

Fig. 2 shows the architecture of integrating DTOML into an MEC system. User level encompasses diverse user equipment (UE), edge level involves MEC provides edge computing services, with edge hosts offering services and cloud servers at the remote level, as detailed below:

(1) At the user layer: the parser operating at the user level, transforms applications into a DAG. The local trainer handles the "inner loop" training by receiving the parsed DAG as training data from the parser. The trained policy network parameters are transmitted to the MEC

host. After the training process is completed, the uninstallation scheduler makes uninstallation decisions based on policy network inference. On the other hand, any tasks that are offloaded are transmitted to the MEC host for further processing.

(2) At the edge layer: The MEC platform utilizes modules for global training services and remote execution services. The global training service handles "outer-loop" training by exchanging policy network parameters with User Equipment (UE). The remote execution service manages tasks offloaded from the UE, assigns them to the appropriate cloud, and returns the results to the UE.

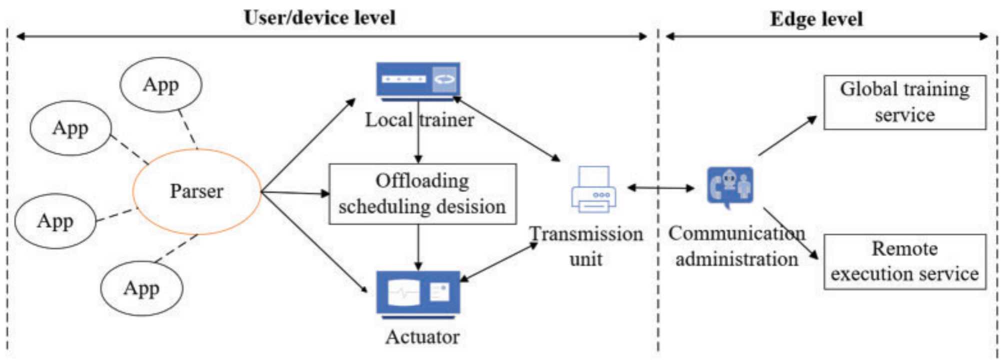


Figure 2: Edge computing system architecture

Next, we'll outline the DTOML algorithm's training process in the MEC system, depicted in Fig. 3. This process comprises four steps:

(1) Initial parameter download: UE retrieves meta-policy parameters from MEC host.

(2) UE-side inner-loop training: Performing "inner-loop" training on each UE using meta-strategy and local data to acquire task-specific strategy. Simultaneously, UE uploads task-specific strategy parameters to the MEC host.

(3) MEC-side outer-loop training: MEC host conducts "outer-loop" training based on collected task-specific strategy parameters, generates a new meta-strategy, and initiates a new training round. Once a stable meta-strategy is achieved, it learns the task-specific strategy of new UEs through "inner-loop" training. Algorithm details will be explained in Section 4.3.

(4) Target optimization: Based on the specific task strategy of the learned new UE, use the NSGA-II algorithm to jointly optimize the delay and energy consumption.

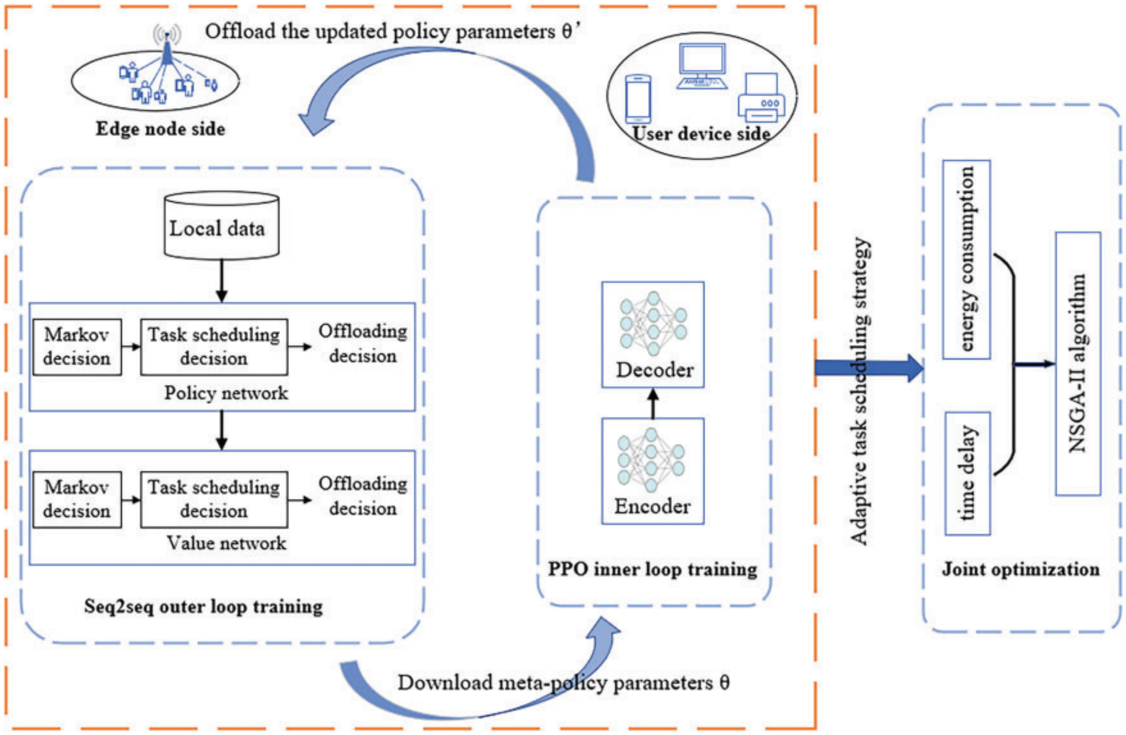


Figure 3: The algorithm structure diagram of the design

4.2 Multiple MDP Modeling

To address the computation offloading challenge within diverse MEC environments, in this paper, we formulat the computation offloading process as multiple MDPs. The overall learning process is bifurcated into two segments: efficaciously acquiring meta-strategies across all MDPs and rapidly learning a strategy for a MDP by leveraging meta-strategies. The definitions of each element in MDPs are as follows:

(1) State S: When scheduling task $a_{i,k}$, the task delay is critically dependent on several factors, including the task configuration file (which encompasses the necessary CPU cycles and data size), the DAG topology, the transmission rate, and MEC resources. The MEC resource status, in particular, plays a pivotal role in the offloading decisions made prior to time t_i. Thus, we characterize the status by combining the coding DAG with offloading plan: $S = \{s_{i,k} | s_{i,k} = (G(T,E), A_{1:i,k})\}$, among them, $G(T,E)$ consists of task embedding sequence, and $A_{1:i,k})$ is the offloading plan of the first i tasks and k subtasks.

(2) Action A: Task scheduling involves making binary decisions, defining the action space as $A := \{0,1\}$. In this context, the 0 indicates that the task will be executed on the UE, while the 1 signifies that the task will be offloaded.

(3) Reward R: The reward function is defined as: $\Delta T^c_{i,k} = T^c_{A_{1:i,k}} - T^c_{A_{1:i-1,k}}$.

4.3 Inner Ring Training and Outer Ring Training

DTOML and gradient-based MRL algorithm share similar algorithm structure, and the scheme consists of two training cycles. The objective function is defined based on near-end strategy optimization (PPO). Compared with other strategy gradient methods, PPO has better exploration ability and training stability. Initially, each sampled learning task undergoes training as an "inner loop." Following the completion of the "inner loop" training, the meta-strategy parameter θ is updated using the gradient ascent method through Adam (Adaptive Moment Estimation) to achieve outer loop training. In this scheme, the inner loop training and outer loop training complement each other to update parameters and obtain the optimal strategy. The inner loop training is mainly based on meta policies and local data to run "inner loop" training on each UE to obtain task specific policies, while the outer loop training is to perform "outer loop" training on the collected task specific policy parameters, generate new meta policies, pass them back to the inner loop, and start a new round of training until they are optimal.

4.3.1 "Inner Ring" Training Based on PPO Strategy Gradient Method

For a learning task $a_{i,k}$, initialize the policy parameters $\theta_{i,k}$ and target policy $\pi_{\theta_{i,k}}$, generate trajectory $\tau_{i,k}$ through sample policy $\pi_{\theta^o_{i,k}}$ sampling, then calculate the optimization objective function $J^{PPO}_{\tau_{i,k}}(\theta_{i,k})$, and update the policy parameters $\theta_{i,k}$ through the optimization objective function $J^{PPO}_{\tau_{i,k}}(\theta_{i,k})$. During multiple periods, update the sample policy sampling $\pi_{\theta^o_{i,k}}$ to avoid large-scale updates, and finally repeat the training until convergence, achieving optimization of task specific policies. The objective function for each inner loop training based on PPO is defined as the follows:

$$J^{PPO}_{\tau_{i,k}}(\theta_{i,k}) = J^{C}_{\tau_{i,k}}(\theta_{i,k}) - c_1 J^{VF}_{\tau_{i,k}}(\theta_{i,k}) \tag{24}$$

where the objective function of the PPO algorithm is denoted as $J^{PPO}_{\tau_{i,k}}(\theta_{i,k})$, the optimization objectives is denoted as $J^{C}_{\tau_{i,k}}(\theta_{i,k})$, the hyperparameters of the value function term in balanced optimization objectives is denoted as c_1, the value function is denoted as $J^{VF}_{\tau_{i,k}}(\theta_{i,k})$, the generated trajectory is denoted as $\tau_{i,k}$, the parameters of the strategy is denoted as $\theta_{i,k}$.

Algorithm 1: Pseudocode for inner loop training based on PPO

Input: Initial policy parameters $\theta_{i,k}$; Sample policy $\pi_{\theta^o_{i,k}}$; Optimization objectives $J^{C}_{\tau_{i,k}}(\theta_{i,k})$; Hyperparameters c_1; Number of trajectories N_T.
Output: Optimized strategy parameters $\theta'_{i,k}$.

```
1: for iterations $t = 1{:}N_T$ do
2:     Sample policy $\pi_{\theta^o_{i,k}}$ generates trajectory $\tau_{i,k}$.
3:     Calculate optimization objectives $J^{C}_{\tau_{i,k}}(\theta_{i,k})$.
4:     Calculatevalue function $J^{VF}_{\tau_{i,k}}(\theta_{i,k})$.
5:     Calculate the optimization objective function:
6:             $J^{PPO}_{\tau_{i,k}}(\theta_{i,k}) = J^{C}_{\tau_{i,k}}(\theta_{i,k}) - c_1 J^{VF}_{\tau_{i,k}}(\theta_{i,k})$
7:     Update policy parameter policy parameters $\theta_{i,k}$.
8:     Output optimized policy parameters $\theta'_{i,k}$.
9: end
10: Policy update.
```

The pseudocode for inner loop training based on PPO is shown in Algorithm 1.

4.3.2 "Outer Loop" Training Based on Seq2seq Network

The seq2sequence network mainly consists of an encoder and a decoder. Additionally, it incorporates an attention mechanism, enabling the decoder to focus on distinct segments of the encoder input sequence during each step of output generation. This helps mitigate information loss inherent in seq2sequence networks.

According to PPO algorithm and the objective function of inner-loop training, the objective function of each outer-loop training is defined as the following formula:

$$J^{DTOML}(\theta_{i,k}) = E_{a_{i,k}\sim\rho(a_{i,k}),\pi\sim P_{\tau_i}(\tau,\theta'_{i,k})}. \tag{25}$$

where the objective function of the outer loop is denoted by $J^{DTOML}(\theta_{i,k})$, the expected value under the given learning task, task sampling distribution and policy distribution is denoted as $E_{a_{i,k}\sim\rho(a_{i,k}),\pi\sim P_{\tau_i}(\tau,\theta'_{i,k})}$, the sampling distribution of learning task $a_{i,k}$ is denoted as $\rho(a_{i,k})$, the strategy for learning tasks is denoted as π, the strategy distribution for given trajectory τ and optimized strategy parameters $\theta'_{i,k}$ is denoted as $P_{\tau_i}(\tau,\theta'_{i,k})$.

The network structure diagram of seq2seq is shown in Fig. 4. The seq2seq network mainly includes the following parts: (1) Input sequence encoding: The encoder is composed of an RNN, which is used to gradually process the input task sequence and finally output a context vector that compresses the information of the input sequence. (2) Context vector generation: The final hidden state of the encoder is used as a context vector to represent the features of the input sequence. This vector will be used in the decoder. (3) Output sequence decoding: The context vector is passed into the decoder, which is also an RNN that gradually generates the output sequence. The decoder generates an output at each step, which is the offloading decision for each task.

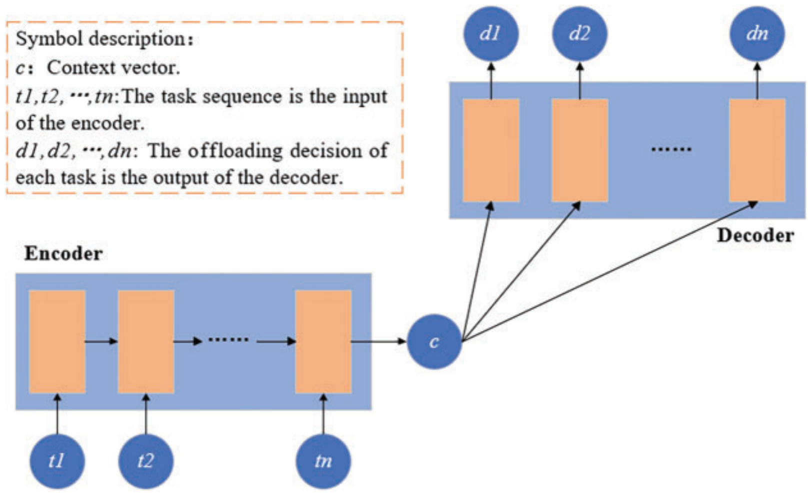

Symbol description:
c: Context vector.
t1,t2, ⋯,tn: The task sequence is the input of the encoder.
d1,d2, ⋯,dn: The offloading decision of each task is the output of the decoder.
d1
d2
dn
Decoder
Encoder
c
t1
t2
tn

Figure 4: Network structure diagram of seq2seq

4.4 Joint Optimization Algorithm Based on NSGA-II Algorithm

The NSGA-II algorithm has the advantages of efficiently handling multi-objective problems and maintaining diversity of solutions. It can not only find Pareto optimal solutions through non dominated sorting, but also balance multiple objectives simultaneously. We also use the crowding distance mechanism to maintain the diversity of solutions on the Pareto front. This ensures that the optimization process can explore a wide range of potential solutions, rather than just converging to a single solution point. Therefore, this algorithm was chosen for further optimization. Using the time delay and energy consumption during task scheduling with the adaptive scheduling strategy as input for the NSGA-II algorithm, efficient joint optimization results can be swiftly achieved through operations like selection, crossover, mutation, and rapid non-dominated sorting. The NSGA-II algorithm's joint optimization process unfolds as follows:

Algorithm 2: NSGA-II algorithm pseudocode

```
Input: Cost function $MOP2$; Population size $N$; Maximum number of Iterations $maxIt$; Crossover probability $P_c$; Mutation probability $P_m$.
Output: Total target value $Q$.
 1: Initialize population $P_t$ and individual fitness $fit$.
 2:   for iterations $t_s < maxIt$ do
 3:     Create an empty offspring population $I(t)$.
 4:     for $|I(t)| \leq N$
 5:        Select parent individuals from $P_t$
 6:        Perform cross operation with probability $P_c$ and generate offspring individuals
 7:        Perform mutation operation with probability $P_m$
 8:        Evaluate the fitness of offspring individuals
 9:        Add offspring individuals to $I(t)$.
10:     end
11:     Form pareto dominance sorting $R(t)$.
12:     Selecting the top $N$ individuals in $R(t)$ based on crowding degree.
13:     The Pareto front solutions from the final generation.
14:   end.
```

(1) Initially creating a population of size N using random generation, followed by non-dominant sorting.
(2) From the second generation onwards, merging the parent population and offspring population for efficient non-dominant sorting.
(3) Generate new offspring population through the basic operation of genetic algorithm, until the constraint condition of the algorithm is met, that is, the iteration number of the algorithm it is less than the maximum iteration number $Maxit$.

Algorithm 2 is pseudocode for the NSGA-II algorithm.

5 Performance Evaluation

Basic Environment Setting

This paper is conducted in Python 3.7 simulation environment. The number of users is 10–50, the effective distance of wireless communication is 60–100 m, the data size of a single task is 2000–10,000 kB, the CPU cycle provided to each user is 8–12 MHz, and the transmission rate is determined by Shannon formula.

Fig. 5 shows the variation of task offloading latency and energy consumption with the number of gradient update steps in a 2×2 network environment (network environment A) with five algorithm schemes. Fig. 5a can be observed that during the entire gradient step size update process, the proposed DTOML scheme has lower task offloading latency than the Greedy scheme [16], RL scheme [27], and

DRL scheme [16] in a 2 × 2 network environment. Fig. 5b can be observed that the proposed DTOML scheme has lower task offloading energy consumption than the Greedy scheme [16], RL scheme [27], PSO scheme, and DRL scheme [16] in a 2 × 2 network environment throughout the entire gradient step size update process. Due to the fact that the Greedy, RL, PSO, DRL, and DTOML schemes update their strategies through parameter optimization iterations with increasing gradient update steps, the poor performance of the Greedy algorithm results in a stable trend in the target value with increasing gradient steps. In contrast, the target value in the RL scheme, DRL scheme, and DTOML scheme all show a decreasing trend. DTOML scheme and DRL scheme are obviously superior to Greedy scheme, PSO scheme and RL scheme, because both of them are model learning and updating strategies based on pre training, but the optimization effect of DTOML algorithm is the best.

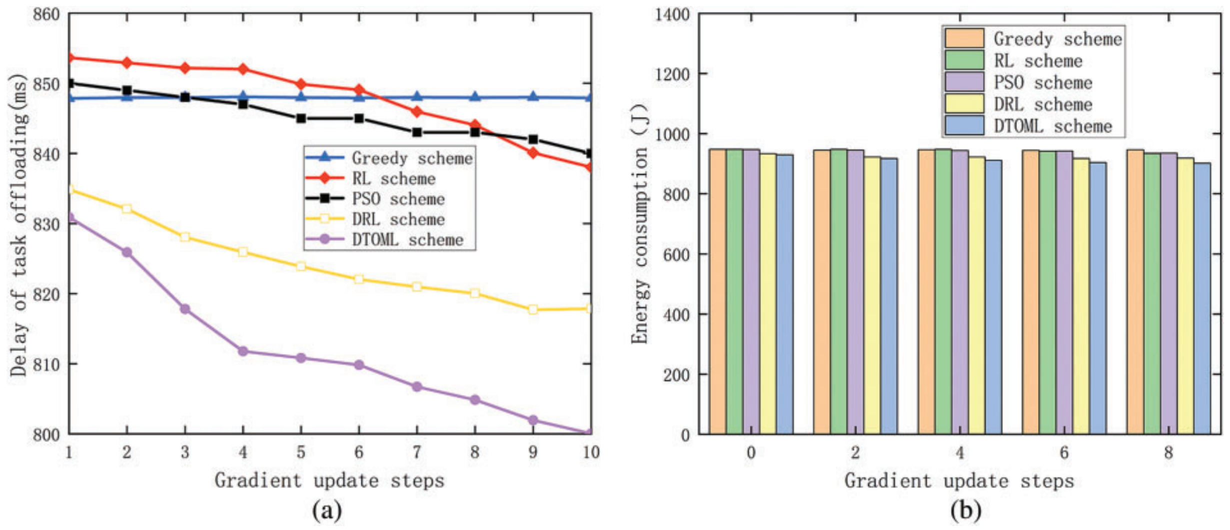


Figure 5: The impact of gradient update steps on task offloading latency and energy consumption in a 2 × 2 network environment (network environment A). (a) Latency. (b) Energy consumption

Fig. 6 shows the variation of task offloading latency and energy consumption with gradient update steps for five algorithm schemes in a 4 × 4 network environment (network environment B). From Fig. 6a, it can be observed that the proposed DTOML scheme has lower task offloading latency than the Greedy scheme and RL scheme in a 4 × 4 network environment throughout the entire gradient update process. When the gradient update step size is small, there is almost no difference in task offloading latency between the DTOML scheme and the DRL scheme. As the gradient update step size increases, the task offloading latency of the DTOML scheme is lower than that of the DRL scheme, exhibiting better performance. From Fig. 6b, it can be observed that the proposed DTOML scheme has lower task offloading energy consumption than the Greedy scheme, RL scheme, and DRL scheme in a 4 × 4 network environment throughout the entire gradient update process. Due to the fact that the Greedy scheme, PSO scheme, RL scheme, DRL scheme, and DTOML scheme all update their strategies through parameter optimization iterations as the gradient update step size increases, the poor performance of the Greedy scheme results in a stable trend in target value as the gradient step size is updated. On the other hand, the RL scheme and DTOML scheme show a decreasing trend in target value, while the DRL scheme performs worse as it changes from a 2 × 2 network environment to a more complex 4 × 4 network environment, causing the trend of task offloading latency to fluctuate up and down. The DTOML scheme and DRL scheme are obviously superior to

Greedy scheme, PSO scheme and RL scheme, because both DTOML and DRL are model learning and updating strategies based on pre training, but DTOML has the best optimization effect, because DTOML scheme achieves the results that target value are lower than Greedy scheme, RL scheme, PSO scheme and DRL scheme by optimizing parameter update efficiency, collaborative learning effect and network transmission overhead.

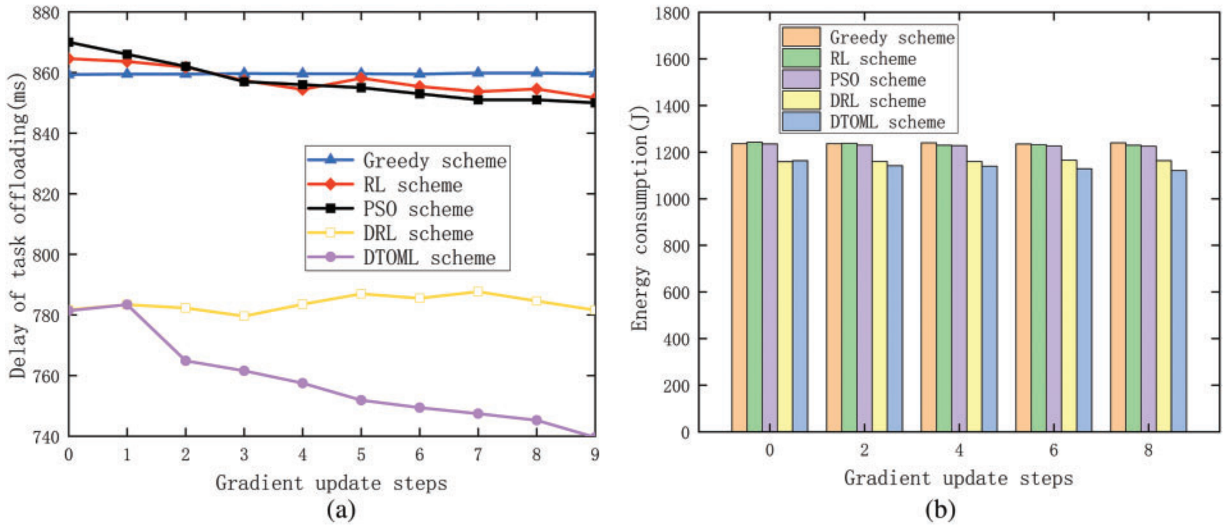


Figure 6: The impact of gradient update steps on task offloading latency and energy consumption in a 4 × 4 network environment (network environment A). (a) Latency. (b) Energy consumption

Fig. 7 shows the variation of task offloading latency and energy consumption with gradient update steps for five algorithm schemes in a 3 × 3 network environment (network environment C). From Fig. 7a, it can be observed that during the entire gradient update process, the proposed DTOML scheme has a lower task offloading latency than the Greedy scheme, PSO scheme and RL scheme in a 3 × 3 network environment. When the gradient update step size is small, the task offloading delay of the DTOML scheme is higher than that of the DRL scheme, and as the gradient update step size increases, the task offloading delay of the DTOML scheme is lower than that of the DRL scheme, exhibiting better performance. From Fig. 7b, it can be observed that during the entire gradient update process, the proposed DTOML scheme has lower task offloading energy consumption than the Greedy scheme, PSO scheme, RL scheme, and DRL scheme in a 3 × 3 network environment. When the gradient update step size is small, the task offloading energy consumption of the DTOML scheme is higher than that of the DRL scheme, and as the gradient update step size increases, the task offloading energy consumption of the DTOML scheme is lower than that of the DRL scheme, exhibiting better performance.

Due to the fact that the Greedy scheme, RL scheme, PSO scheme, DRL scheme, and DTOML scheme update their strategies through parameter optimization iterations with increasing gradient update step size, the performance of the Greedy algorithm is poor as the gradient step size is updated, resulting in a stable trend in target value. When the network environment changes from 4 × 4 to 3 × 3, the performance of the DRL scheme improves, causing the trend of task offloading latency to fluctuate up and down. The DTOML scheme and DRL scheme are obviously superior to Greedy scheme and RL scheme, because both DTOML and DRL are model learning and updating strategies based on pre training, but the optimization effect of DTOML is the best, because DTOML scheme achieves the effect that target value are lower than Greedy scheme, RL scheme and DRL scheme by optimizing parameter update efficiency, collaborative learning effect and network transmission overhead. The three different network environments A, B, and C are formed by the dynamic changes in the network environment caused by the entry and exit of edge devices. According to Figs. 5–7, it can be observed that the proposed DTOML scheme has lower task offloading latency and energy consumption compared to the Greedy scheme, RL scheme, PSO scheme and DRL scheme. This indicates that the proposed DTOML scheme always ensures better adaptability in different network environments (A, B, C).

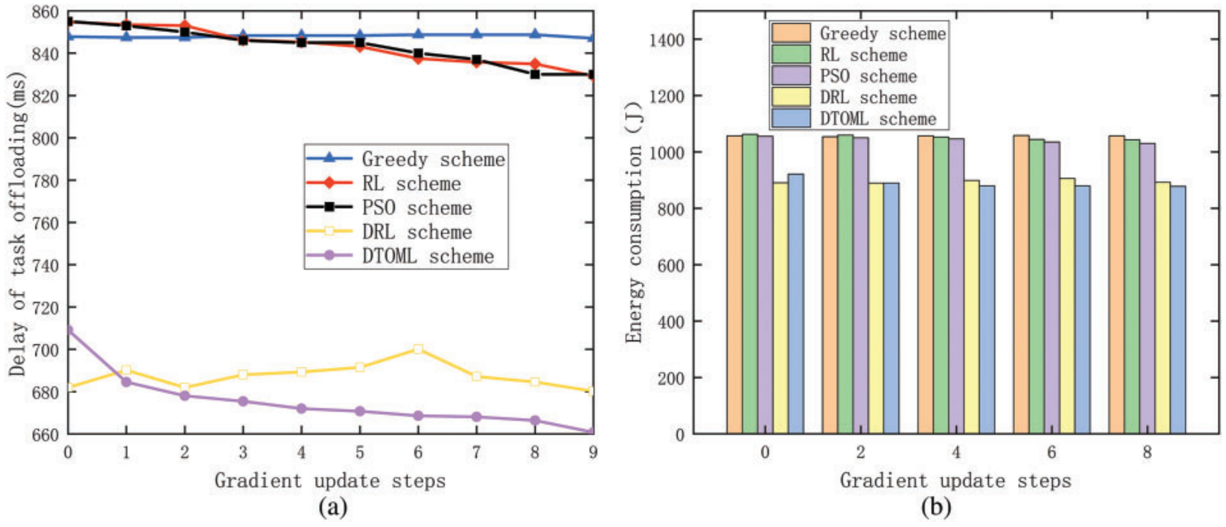


Figure 7: The impact of gradient update steps on task offloading latency and energy consumption in a 3 × 3 network environment (network environment A). (a) Latency. (b) Energy consumption

5.1.1 Influence of Transmission Rate

Learning the offloading strategy for each transmission rate is considered a separate learning task, and the trained meta strategy is evaluated at transmission rates of 6, 9, and 12 Mbps. From Fig. 8a,c,e, it can be observed that as the channel transmission rate increases, the overall task offloading delay shows a downward trend. That is, when the rate is 9 Mbps, the task offloading delay is generally smaller than when the rate is 6 Mbps, and when the rate is 12 Mbps, the task offloading delay is basically smaller than when the rate is 9 Mbps. As shown in Fig. 8a, when the rate is 6 Mbps, it can be seen from the trend of the task offloading delay of the five algorithm schemes with the number of gradient update steps that the proposed DTOML scheme has a higher task offloading delay than the DRL scheme at the initial gradient update step, and a lower task offloading delay than the Greedy scheme, RL scheme, and DRL scheme at other gradient update steps. From Fig. 8c,e, it can be observed that when the channel transmission rate increases from 6 to 9 and 12 Mbps, even in the initial update steps, the DTOML scheme achieves the lowest task offloading delay among the five algorithm schemes. This proves that the proposed DTOML scheme has better delay optimization ability than the Greedy scheme, PSO scheme, RL scheme, and DRL scheme.

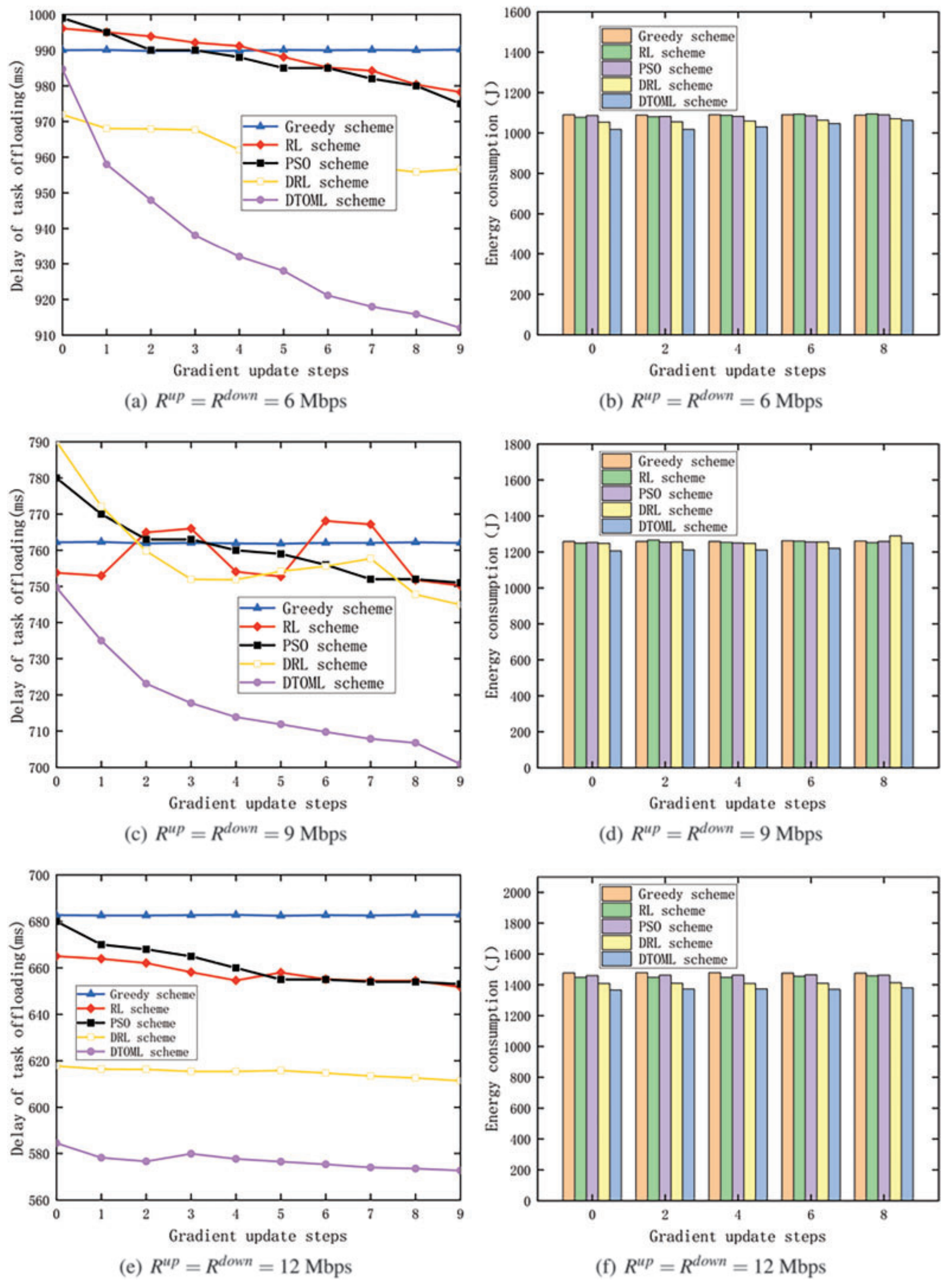


Figure 8: The influence of different transmission rates on task offload delay and energy consumption

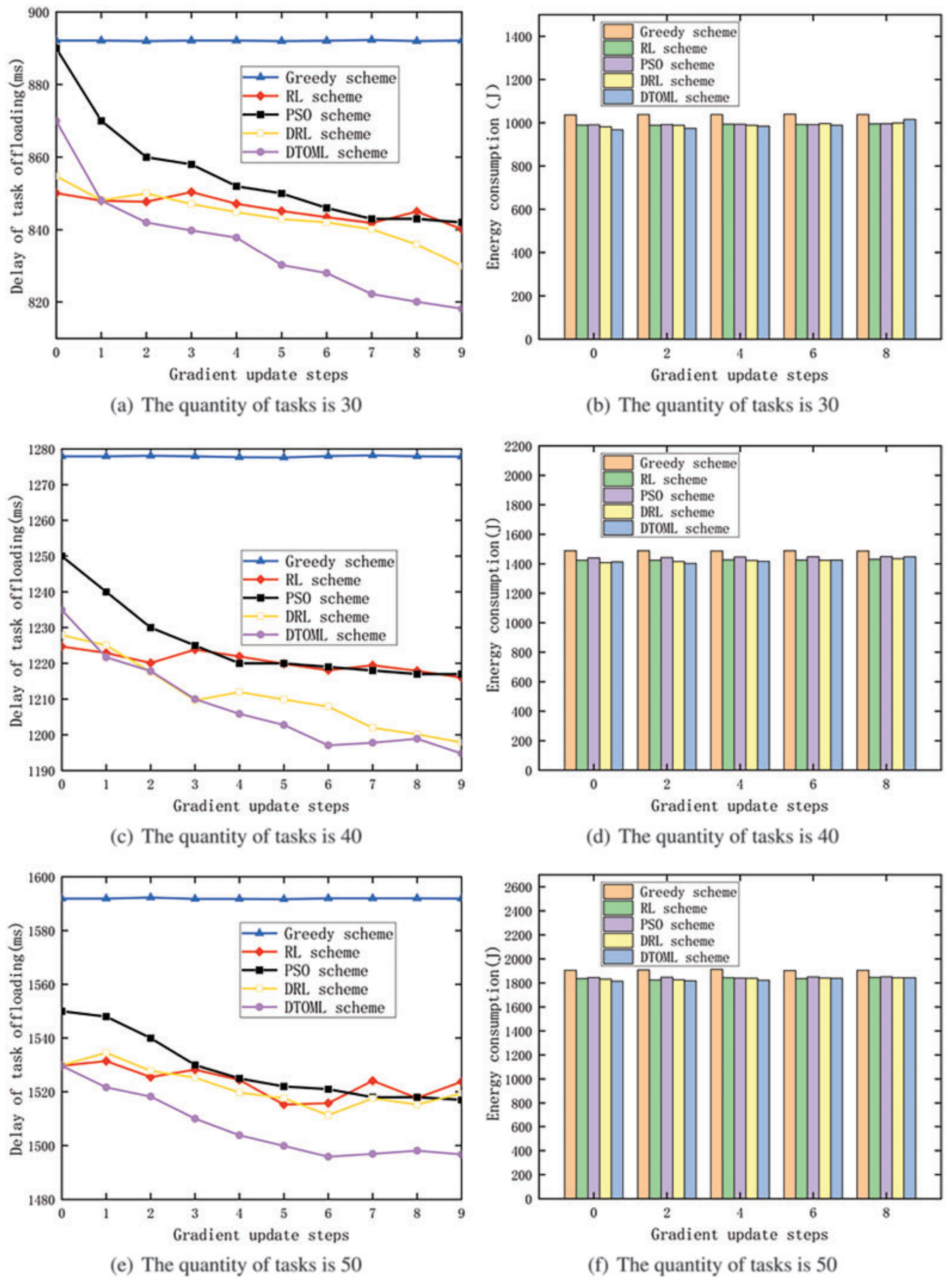
5.1.2 Impact of the Number of Tasks

From Fig. 8b,d,f, it can be observed that as the channel transmission rate increases, the overall energy consumption of task offloading shows an upward trend. That is, when the channel transmission rate is 9 Mbps, the energy consumption of task offloading is generally higher than that when the channel transmission rate is 6 Mbps, and when the channel transmission rate is 12 Mbps, the energy consumption of task offloading is basically higher than that when the channel transmission rate is 9 Mbps. The trend of task offloading energy consumption with gradient step updates for the five algorithm schemes shows that although the task offloading energy consumption increases with the increase of gradient update steps, the proposed DTOML scheme consistently has lower task offloading energy consumption than the Greedy scheme, RL scheme, PSO scheme and DRL scheme, demonstrating good performance. From Fig. 8d,f, it can be observed that when the channel transmission rate increases from 6 to 9 and 12 Mbps, the DTOML scheme also has the lowest task offloading energy consumption among the five algorithm schemes. This proves that the proposed DTOML scheme has better energy optimization ability than the Greedy scheme, RL scheme, PSO scheme and DRL scheme.

Fig. 9 shows the variation of task offloading latency and energy consumption with gradient update steps for five algorithm schemes under different task quantities. From Fig. 9a,c,e, it can be observed that as the number of tasks increases, the overall trend of task offloading delay shows an increasing trend, that is, the task offloading delay when the quantity is 40 is greater than that when the quantity is 30, and the task offloading delay when the quantity is 50 is greater than that when the quantity is 40. As shown in Fig. 9a, the trend of task offloading delay of the five algorithm schemes with the update of gradient steps can be observed that during the entire update process of gradient steps, the DTOML scheme proposed has a lower task offloading delay than the Greedy scheme when the quantity is 30. The DTOML scheme proposed has a higher task offloading delay than the DRL scheme at the initial gradient update steps, but at other gradient update steps, the DTOML scheme has a lower task offloading delay than the Greedy scheme, RL scheme, PSO scheme and DRL scheme. From Fig. 9a,c,e, it can be observed that when the quantity increases from 30 to 50, the task offloading delay of the DTOML scheme is only slightly higher than the other three schemes at the initial update step. As the gradient update step increases, the task offloading delay of the DTOML scheme is lower than the other three schemes, proving that the proposed DTOML scheme has better delay optimization ability than the Greedy scheme, RL scheme, PSO scheme and DRL scheme.

From Fig. 9b,d,f, it can be observed that as the number of tasks increases, the overall energy consumption of task offloading shows an upward trend. The trend of task offloading energy consumption with gradient step updates for the five algorithm schemes shows that although the task offloading energy consumption increases with the increase of gradient update steps, the proposed DTOML scheme has overall lower task offloading energy consumption than the Greedy scheme, RL scheme, and DRL scheme, demonstrating good performance. From Fig. 9d,f, it can be observed that when the quantity increases from 30 to 40 and 50, the DTOML scheme is also the scheme with the lowest overall task offloading energy consumption among the five algorithm schemes. This proves that the proposed DTOML scheme has better energy optimization ability than the Greedy scheme, RL scheme, PSO scheme and DRL scheme.

The scheme proposed in this article obtains task specific strategies through inner loop training of PPO algorithm. Compared with reinforcement learning, particle swarm optimization and other algorithms, it has the advantages of handling high-dimensional state space and adapting to dynamic environments that reinforcement learning, particle swarm optimization and other algorithms do not have. In addition, this scheme models multiple Markov decision processes (MDPs), processes sequence data through seq2seq network, and uses NSGA-II algorithm for Pareto optimization, which can more comprehensively capture the dynamic features of different tasks and environments. It has stronger accuracy, effectiveness, adaptability, and flexibility than deep reinforcement learning algorithms, and can better reduce energy consumption and latency while ensuring performance, and improve overall system efficiency.



(a) The quantity of tasks is 30 (b) The quantity of tasks is 30

(c) The quantity of tasks is 40 (d) The quantity of tasks is 40

(e) The quantity of tasks is 50 (f) The quantity of tasks is 50

Figure 9: The impact of different task quantities on task offloading latency and energy consumption

Tables 2 and 3 compare the average task offloading latency and energy consumption of five algorithm schemes under different network environments, channel transmission rates, and task volumes.

It can be observed that under different comparison indicators, the proposed DTOML scheme has lower average latency and energy consumption compared to the Greedy scheme, RL scheme, and DRL scheme. In different network environments, the lower the average latency and energy consumption of task offloading in the DTOML scheme, the better its adaptability. At different channel transmission rates and task volumes, the lower the average task offloading delay and average energy consumption of the DTOML scheme, the better the joint optimization ability.

Table 2: The average time delay of five algorithm schemes under different conditions

Data	Greedy scheme/(ms)	RL scheme/(ms)	PSO scheme/(ms)	DRL scheme/(ms)	DTOML scheme/(ms)
Network environment A	848.0	847.8	845.2	824.4	812.2
Network environment B	848.0	843.0	843.1	687.5	675.2
Network environment C	860.0	857.9	857.1	784.2	758.7
$R^{up} = R^{down} = 6$	990.0	988.3	986.9	962.8	935.6
$R^{up} = R^{down} = 9$	762.0	758.2	760.6	758.7	717.8
$R^{up} = R^{down} = 12$	683.0	657.9	661.4	615.4	577.4
Task number = 30	892.0	845.8	855.4	843.6	835.6
Task number = 40	1278.0	1220.6	1225.6	1211.1	1208.3
Task number = 50	1592.0	1523.9	1528.9	1522.2	1507.2
CPU cycles	1118.2	1049.1	1052.3	966.4	930.9

Table 3: The average energy consumption of five algorithm schemes under different conditions

Data	Greedy scheme/(J)	RL scheme/(J)	PSO scheme/(ms)	DRL scheme/(J)	DTOML scheme/(J)
Network environment A	948.0	948.4	942.6	925.2	913.8
Network environment B	1050.8	1054.6	1043.2	897.4	888.8
Network environment C	1240.0	1238.6	1228.8	1164.0	1140.2
$R^{up} = R^{down} = 6$	1090.0	1089.2	1085.2	1063.2	1036.4
$R^{up} = R^{down} = 9$	1262.0	1258.6	1254.4	1261.2	1220.8
$R^{up} = R^{down} = 12$	1483.0	1458.6	1462.4	1415.8	1378.0
Task number = 30	1042.0	996.6	992.4	995.6	989.6
Task number = 40	1488.0	1430.6	1444.6	1423.2	1421.0

(Continued)

Table 3 (continued)

Data	Greedy scheme/(J)	RL scheme/(J)	PSO scheme/(ms)	DRL scheme/(J)	DTOML scheme/(J)
Task number = 50	1912.0	1843.0	1846.2	1841.2	1829.2
CPU cycles	1460.6	1423.2	1420.8	1395.4	1396.3

From Fig. 10, it can be observed that as the number of CPU cycles allocated to each user increases, the time delay and energy consumption of each algorithm gradually decrease, but the decrease in time delay will be more pronounced. This is because as the allocated CPU cycles increase, there are more available computing resources, and tasks are processed faster, resulting in an obvious reduction in time delay. The system energy consumption includes two parts: transmission and computation energy consumption. Although more computing resources are allocated, the task offloading process involves two processes: transmission and computation, both of which consume energy. Therefore, the reduction in energy consumption is not as significant.

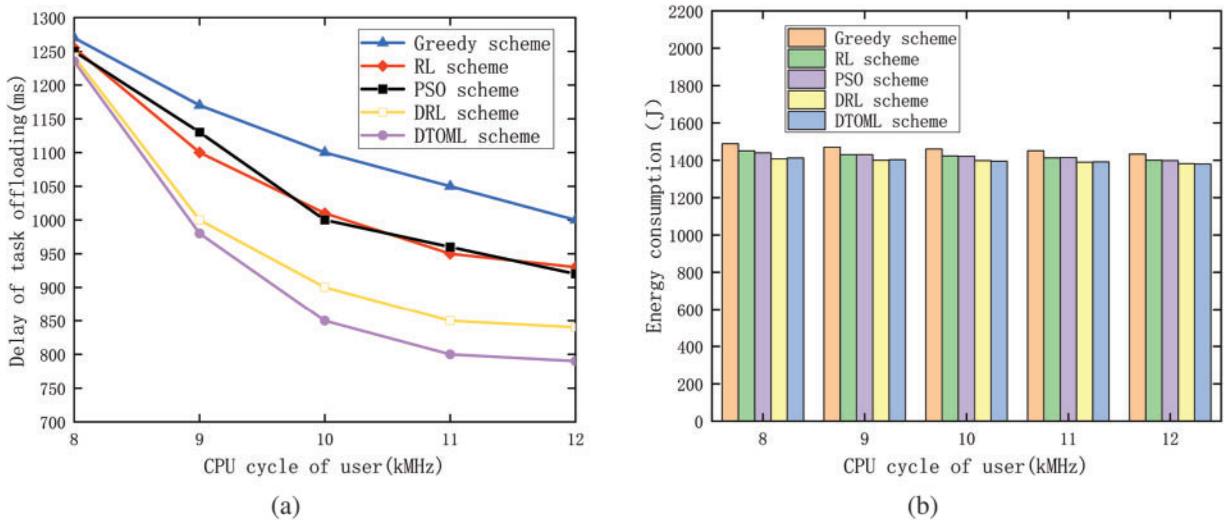


Figure 10: The impact of the CPU cycle for each user on task offloading latency and energy consumption. (a) Latency. (b) Energy consumption

6 Conclusion

In this paper, to solve the problem that the learning task offloading strategy cannot have good adaptability in different network environments, a distributed edge computing adaptive task offloading scheme based on MRL is proposed, which simultaneously realizes the joint optimization of delay and energy consumption in edge computing networks. In this paper, we first establish various network environment models and formulate the task offloading strategy using DAGs. Subsequently, we employ meta reinforcement learning MDP and seq2seq for adaptive training of task scheduling strategies. To jointly optimize delay and energy consumption, we integrate the NSGA-II algorithm. Simulation results reveal that the proposed distributed edge computing adaptive task offloading scheme (DTOML), which utilizes a meta strategy, exhibits remarkable advantages across various network environments, transmission rates, and task sequences. From the average values in Tables 2 and 3, it can be observed that our algorithm performs the best. In comparison to other edge computing adaptive task offloading schemes, the DTOML achieves a 21% reduction in delay and a 19% reduction in energy consumption.

For more dynamic or resource constrained environments, the number and layers of neurons in the PPO actor network and critic network during inner loop training can be increased to enhance the model's expressive power. In the future, multi-agent collaborative offloading strategy will be further studied to enable multiple edge computing nodes to work in coordination and further improve the efficiency and robustness of the overall system.

Acknowledgement: The authors gratefully acknowledge the helpful comments and suggestions of the reviewers and editors, which have improved the presentation.

Funding Statement: This research was funded by the Fundamental Research Funds for the Central Universities (J2023-024, J2023-027).

Author Contributions: Conceptualization, Jiajia Liu and Jianhua Liu; methodology, Jiajia Liu and Jianhua Liu; validation, Jiajia Liu, Peng Xie, Wei Li, Bo Tang and Jianhua Liu; investigation, Jiajia Liu, Peng Xie and Wei Li; writing—original draft preparation, Wei Li and Jianhua Liu; writing—review and editing, Jiajia Liu, Peng Xie, Wei Li, Bo Tang and Jianhua Liu; supervision, Jiajia Liu and Jianhua Liu; project administration, Jianhua Liu. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the corresponding author, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

[1] M. -R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," *Commun. Biol.*, pp. 43–56, 2011. doi: 10.1145/1999995.2000000.

[2] Z. Liu *et al.*, "PPRU: A privacy-preserving reputation updating scheme for cloud-assisted vehicular networks," *IEEE Trans. Vehicular Technol.*, pp. 1–16, 2023. doi: 10.1109/TVT.2023.3340723.

[3] D. Sabella *et al.*, "Developing software for multi-access edge computing," *ETSI White Paper*, vol. 20, pp. 1–38, 2019. Accessed: Aug. 26, 2024. [Online]. Available: https://www.etsi.org.

[4] J. B. da Costa *et al.*, "Mobility-aware vehicular cloud formation mechanism for vehicular edge computing environments," *Ad Hoc Netw.*, vol. 151, no. 1, 2023, Art. no. 103300. doi: 10.1016/j.adhoc.2023.103300.

[5] Z. Liu *et al.*, "PPTM: A privacy-preserving trust management scheme for emergency message dissemination in space–air–ground-integrated vehicular networks," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 5943–5956, 2022. doi: 10.1109/JIOT.2021.3060751.

[6] Q. Tang, C. Dai, Z. Yu, D. Cao, and J. Wang, "An UAV and EV based mobile edge computing system for total delay minimization," *Comput. Commun.*, vol. 212, no. 1, pp. 104–115, 2023. doi: 10.1016/j.comcom.2023.09.025.

[7] J. Liu, Z. Wu, J. Liu, and X. Tu, "Distributed location-aware task offloading in multi-UAVs enabled edge computing," *IEEE Access*, vol. 10, no. 2, pp. 72416–72428, 2022. doi: 10.1109/ACCESS.2022.3189682.

[8] J. Liu, P. Xie, J. Liu, and X. Tu, “Task offloading and trajectory optimization in UAV networks: A deep reinforcement learning method based on SAC and A-star,” *Comput. Model. Eng. Sci.*, vol. 141, no. 2, pp. 1243–1273, 2024. doi: 10.32604/cmes.2024.054002.

[9] J. Liu, Z. Wu, J. Liu, and Y. Zou, “Cost research of internet of things service architecture for random mobile users based on edge computing,” *Int. J. Web Inf. Syst.*, vol. 18, no. 4, pp. 217–235, 2022. doi: 10.1108/IJWIS-02-2022-0039.

[10] X. Lin, Y. Wang, Q. Xie, and M. Pedram, “Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment,” *IEEE Trans. Serv. Comput.*, vol. 8, no. 2, pp. 175–186, 2015. doi: 10.1109/TSC.2014.2381227.

[11] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji and M. Bennis, “Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, 2019. doi: 10.1109/JIOT.2018.2876279.

[12] G. Qu, H. Wu, R. Li, and P. Jiao, “DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing,” *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 3, pp. 3448–3459, 2021. doi: 10.1109/TNSM.2021.3087258.

[13] S. Dong *et al.*, “Task offloading strategies for mobile edge computing: A survey,” *Comput. Netw.*, vol. 254, no. 6, 2024, Art. no. 110791. doi: 10.1016/j.comnet.2024.110791.

[14] A. Al-Shuwaili and O. Simeone, “Energy-efficient resource allocation for mobile edge computing-based augmented reality applications,” *IEEE Wirel. Commun. Lett.*, vol. 6, no. 3, pp. 398–401, 2017. doi: 10.1109/LWC.2017.2696539.

[15] T. Lyu, H. Xu, and Z. Han, “Multi-leader multi-follower stackelberg game based resource allocation in multi-access edge computing,” in *ICC 2022–IEEE Int. Conf. Commun.*, 2022, pp. 4306–4311. doi: 10.1109/ICC45855.2022.9838425.

[16] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, “Fast adaptive task offloading in edge computing based on meta reinforcement learning,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, 2021. doi: 10.1109/TPDS.2020.3014896.

[17] B. Hu, Y. Shi, and Z. Cao, “Adaptive energy-minimized scheduling of real-time applications in vehicular edge computing,” *IEEE Trans. Ind. Inform.*, vol. 19, no. 5, pp. 6895–6906, 2023. doi: 10.1109/TII.2022.3207754.

[18] S. Dong, Y. Xia, and J. Kamruzzaman, “Quantum particle swarm optimization for task offloading in mobile edge computing,” *IEEE Trans. Ind. Inform.*, vol. 19, no. 8, pp. 9113–9122, 2023. doi: 10.1109/TII.2022.3225313.

[19] J. Liu, J. Wei, R. Luo, G. Yuan, J. Liu and X. Tu, “Computation offloading in edge computing for internet of vehicles via game theory,” *Comput. Mater. Contin.*, vol. 81, no. 1, pp. 1337–1361, 2024. doi: 10.32604/cmc.2024.056286.

[20] S. R. Jeremiah, L. T. Yang, and J. H. Park, “Digital twin-assisted resource allocation framework based on edge collaboration for vehicular edge computing,” *Future Gener. Comput. Syst.*, vol. 150, no. 3, pp. 243–254, 2024. doi: 10.1016/j.future.2023.09.001.

[21] H. Wu, W. J. Knottenbelt, and K. Wolter, “An efficient application partitioning algorithm in mobile environments,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1464–1480, 2019. doi: 10.1109/TPDS.2019.2891695.

[22] D. Silver *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. doi: 10.1038/nature24270.

[23] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” 2015. doi: 10.48550/arXiv.1509.02971.

[24] A. Zhu *et al.*, “Computation offloading for workflow in mobile edge computing based on deep Q-learning,” in *2019 28th Wirel. Opt. Commun. Conf. (WOCC)*, 2019, pp. 1–5. doi: 10.1109/WOCC.2019.8770689.

[25] J. Vanschoren, “Meta-learning: A survey,” 2018. doi: 10.48550/arXiv.1810.03548.

[26] H. Fu, H. Tang, J. Hao, W. Liu, and C. Chen, “MGHRL: Meta goal-generation for hierarchical reinforcement learning,” 2019. doi: 10.48550/arXiv.1909.13607.

[27] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell and D. Hassabis, "Reinforcement learning, fast and slow," *Trends Cogn. Sci.*, vol. 23, no. 5, pp. 408–422, 2019. doi: 10.1016/j.tics.2019.02.006.

[28] K. Thar, T. Z. Oo, Z. Han, and C. S. Hong, "Meta-learning-based deep learning model deployment scheme for edge caching," in *2019 15th Int. Conf. Netw. Serv. Manag. (CNSM)*, 2019, pp. 1–6. doi: 10.23919/CNSM46954.2019.9012733.

[29] L. Niu *et al.*, "Multiagent meta-reinforcement learning for optimized task scheduling in heterogeneous edge computing systems," *IEEE Internet Things J.*, vol. 10, no. 12, pp. 10519–10531, 2023. doi: 10.1109/JIOT.2023.3241222.

[30] M. Avgeris, M. Mechennef, A. Leivadeas, and I. Lambadaris, "A two-stage cooperative reinforcement learning scheme for energy-aware computational offloading," in *2023 IEEE 24th Int. Conf. High Performance Switching Routing (HPSR)*, 2023, pp. 179–184. doi: 10.1109/HPSR57248.2023.10147932.

[31] Y. Li, J. Li, Z. Lv, H. Li, Y. Wang and Z. Xu, "GASTO: A fast adaptive graph learning framework for edge computing empowered task offloading," *IEEE Trans. Netw. Serv. Manag.*, vol. 20, no. 2, pp. 932–944, 2023. doi: 10.1109/TNSM.2023.3250395.

[32] H. Xie, D. Ding, L. Zhao, K. Kang, and Q. Liu, "A two-stage preference driven multi-objective evolutionary algorithm for workflow scheduling in the cloud," *Expert. Syst. Appl.*, vol. 238, no. 18, 2024, Art. no. 122009. doi: 10.1016/j.eswa.2023.122009.

[33] P. L. Manikowski, D. J. Walker, and M. J. Craven, "Multi-objective optimisation of the benchmark wind farm layout problem," *J. Mar. Sci. Eng.*, vol. 9, no. 12, 2021. doi: 10.3390/jmse9121376.

[34] N. Panagant, N. Pholdee, S. Bureerat, A. R. Yildiz, and S. Mirjalili, "A comparative study of recent multi-objective metaheuristics for solving constrained truss optimisation problems," *Arch. Computat. Methods Eng.*, vol. 28, no. 5, pp. 4031–4047, 2021. doi: 10.1007/s11831-021-09531-8.

[35] M. Chen and Z. Jun, "Research on layered microgrid operation optimization based on NSGA-II algorithm," in *2018 Int. Conf. Power Syst. Technol. (POWERCON)*, IEEE, 2018, pp. 2149–2156. doi: 10.1109/POWERCON.2018.8602169.

[36] S. Verma, M. Pant, and V. Snasel, "A comprehensive review on NSGA-II for multi-objective combinatorial optimization problems," *IEEE Access*, vol. 9, pp. 57757–57791, 2021. doi: 10.1109/ACCESS.2021.3070634.

[37] P. Wang, F. Xue, H. Li, Z. Cui, L. Xie and J. Chen, "A multi-objective DV-hop localization algorithm based on NSGA-II in internet of things," *Mathematics*, vol. 7, no. 2, 2019, Art. no. 184. doi: 10.3390/math7020184.

[38] D. Liu, Q. Huang, Y. Yang, D. Liu, and X. Wei, "Bi-objective algorithm based on NSGA-II framework to optimize reservoirs operation," *J. Hydrol.*, vol. 585, no. 3, 2020, Art. no. 124830. doi: 10.1016/j.jhydrol.2020.124830.

[39] S. Long, Y. Zhang, Q. Deng, T. Pei, J. Ouyang and Z. Xia, "An efficient task offloading approach based on multi-objective evolutionary algorithm in cloud-edge collaborative environment," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 2, pp. 645–657, 2022. doi: 10.1109/TNSE.2022.3217085.

[40] L. Pan, X. Liu, Z. Jia, J. Xu, and X. Li, "A multi-objective clustering evolutionary algorithm for multi-workflow computation offloading in mobile edge computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1334–1351, 2021. doi: 10.1109/TCC.2021.3132175.

[41] K. Palasundram, N. Mohd Sharef, K. A. Kasmiran, and A. Azman, "Enhancements to the sequence-to-sequence-based natural answer generation models," *IEEE Access*, vol. 8, pp. 45738–45752, 2020. doi: 10.1109/ACCESS.2020.2978551.

[42] Z. Masood, R. Gantassi, and Y. A. Choi, "A multi-step time-series clustering-based seq2seq lstm learning for a single household electricity load forecasting," *Energies*, vol. 15, no. 7, 2022, Art. no. 2623. doi: 10.3390/en15072623.

[43] Y. Keneshloo, T. Shi, N. Ramakrishnan, and C. K. Reddy, "Deep reinforcement learning for sequence-to-sequence models," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 7, pp. 2469–2489, 2020. doi: 10.1109/TNNLS.2019.2929141.

[44] H. Ma, Y. Zhang, S. Sun, and Y. Shan, "A comprehensive survey on NSGA-II for multi-objective optimization and applications," *Artif. Intell. Rev.*, vol. 56, no. 12, pp. 15217–15270, 2023. doi: 10.1007/s10462-023-10526-z.