# A Multiagent Meta-Based Task Offloading Strategy for Mobile-Edge Computing

Weichao Ding, Fei Luo, Chunhua Gu, Zhiming Dai, and Haifeng Lu

***Abstract*—Task offloading in mobile-edge computing (MEC) improves the efficacy of mobile devices (MDs) in terms of computing performance, data storage, and energy consumption by offloading computational tasks to edge servers. Efficient task offloading can leverage MEC technology to reduce task processing latency and energy consumption. By integrating the reasoning ability and machine intelligence of the cognitive computing architecture, such as SOAR and ACT-R, reinforcement learning (RL) algorithms have been applied to resolve the task offloading in MEC. To solve the problem that conventional deep RL (DRL) algorithms cannot adapt to dynamic environments, this article proposed a task offloading scheduling strategy which combined multiagent RL and meta-learning. In order to make the two actions of charging time and offloading strategy fully considered at the same time, we implemented a learning network of two agents on an MD. To efficiently train the policy network, we proposed a first-order approximation method based on the clipped surrogate objective. Finally, the experiments are designed with a variety of the number of subtasks, transmission rate, and edge server performance, and the results show that the MRL-based strategy has the overwhelming overall performance and can be quickly applied in various environments with good stability and generalization.**

***Index Terms*—Deep reinforcement learning (DRL), edge task offloading, meta-learning, multiagent.**

## I. Introduction

With the increasing popularity and high-speed of Mobile Internet, numerous mobile devices (MDs) will be deployed in various business scenarios for data collection and collaborative control. Thus, their MD-based applications, such as target identification and environment sensing, possess great commercial value. However, MDs are limited by various factors, such as cost and size, and they usually are limited by weak performance and small battery capacity; therefore, it is difficult to provide long-term stable services for the above-mentioned scenarios. To solve this problem, mobile-edge computing (MEC) provides high-performance computing resources for MDs at the edge of the network, while wireless power transfer (WPT) services enable MDs to receive continuous and stable power supply from the hybrid access point and store energy in their batteries [1].

The applications of MDs in smart scenarios are addressed by combining MEC and WPT. However, this approach faces three challenges in the related scheduling.

1) Choosing the WPT service duration to ensure successful task processing while reducing the total latency.
2) Reasonably deciding whether application data needs to be offloaded to the corresponding edge server for processing to improve the task success rate.
3) Achieving adaptability of scheduling policies. Improving the robustness of the algorithm to new tasks and environments when the MEC environment or task type changes.

If the WPT service duration is too long, despite providing sufficient power to the MD for task offloading or local computing to ensure task processing completion, the excessive latency may exceed the maximum allowable time for the task and; eventually, it may cause degradation of user experience. If the WPT service duration is too short, the MD may not have enough power for task offloading and cause task processing failure [2]. Furthermore, the mobile application can be split and generate directed acyclic graph (DAG) subtasks as data transmission delay. This enables partial offloading and parallel processing of tasks, and effectively reduces computational latency and transmission delay [3]. Therefore, to fully utilize the computing resources of MDs and edge servers and minimize the corresponding task processing latency and energy consumption, we need to design a strategy to decide whether to offload mobile tasks and WPT service duration. This would ensure that MDs have sufficient power to process and transmit the corresponding tasks and improve the quality of service of users.

To resolve the problem of task offloading with multiple DAG types in the MEC, cognitive computing is an effective way, which integrates reasoning ability and machine intelligence [4], [5]. Especially, artificial intelligence (AI) algorithms enable high performance for edge servers [6]. Therefore, this article tries to utilize one basic AI algorithm of the cognitive computing architecture, such as SOAR [7] and ACT-R [8], i.e., reinforcement learning (RL), to resolve the problem. However, the traditional deep RL (DRL) algorithm [9] is time consuming because it needs to train various types of tasks to learn the latest strategy in a new environment. To fix this problem of the traditional DRL, meta-RL (MRL) utilizes inner and outer loops to accelerate the learning rate of new tasks. The outer loop

Manuscript received 14 November 2022; revised 5 January 2023; accepted 12 February 2023. Date of publication 17 February 2023; date of current version 12 February 2024. This work was supported in part by the Shanghai Sailing Program under Grant 20YF1410900; in part by the National Natural Science Foundation under Grant 62276097; and in part by the Project on Shanghai Science and Technology Innovation Action Plan under Grant 22ZR1416500 and Grant 20dz1201400. *(Corresponding authors: Fei Luo; Chunhua Gu.)*

The authors are with the Faculty of School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China (e-mail: luof@ecust.edu.cn; chgu@ecust.edu.cn).

Digital Object Identifier 10.1109/TCDS.2023.3246107

is responsible for learning the commonality between different tasks from a large number of samples and updating the parameters of the inner loop, which further adjusts the parameters through a short training period to quickly adapt to the new task. Therefore, this study proposes a multiagent MRL strategy [10] to solve the task offloading problem in MEC environments, where mobile applications have multiple DAG types and MDs have WPT services. Consequently, the offloading decisions of each subtask need to coordinate multiple actions, such as the offload decision and required WPT service duration. Finally, the proposed method reduces the total latency and energy consumption of this task as much as possible while ensuring a high task success rate. To evaluate the performance of the strategy thoroughly, we consider the following dynamic scenarios: 1) heterogeneous users with personal preferences of mobile applications, which are represented as DAGs with different heights, widths, and task numbers and 2) varying the transmission rates according to the distance between the MDs and MEC servers. The major contributions of this study can be summarized as follows.

1) Proposing a MEC model where mobile applications can be split into multiple DAG types and MDs with WPT services. A task offloading algorithm, namely, MASAC, based on multiagent RL was proposed aiming to minimize the task processing latency and energy consumption of MDs.
2) Based on the proposed MASAC algorithm, combined with meta-learning, a multiagent MRL policy is constructed, which can quickly solve the offloading problem in dynamic scenarios. The multiagent MRL policy achieves high sample efficiency in new scenarios; thus, it enables MDs to run the training process autonomously.
3) Simulation experiments to minimize the energy consumption and latency are constructed, considering the differences between any two DAG tasks in terms of topology, data volume, and computation complexity. The results indicate that the multiagent MRL can converge to a desirable solution with less training, obtain good stability, and generalize well. Furthermore, it achieves better results in terms of energy consumption, latency, and task success rate compared to other algorithms.

The remainder of this article is organized as follows. Related studies are presented in Section II. The MEC and problem models are described in Section III. The proposed task offloading algorithm is presented in Section IV. In Section V, the experimental results of the simulations are presented. Finally, Section VI presents the conclusions of this study.

## II. Related Studies

Current research on task offloading in MEC applications focuses on the environment and algorithms. To solve the limitation of the MD power, combining the WPT technology with MEC provides a novel solution that ensures the continuity of computing services and reduces the overall system overhead. Current research on WPT-based MEC task offloading has attracted much attention. For example, Zhang et al. [11] investigated the problem of sustainable computational offloading in MEC environments and proposed an online rewards-optimal auction method to optimally handle the total amount of long-term rewards for offloading tasks. Yan et al. [12] proposed a low-complexity critic network algorithm to quickly evaluate the offloading decision by analyzing the structure of the optimal solution. Their experimental results demonstrated that for different types of task graphs, the algorithm achieved an optimization performance of 99.1%, while significantly reducing the computational complexity compared to existing optimization methods. Li et al. [13] proposed a dynamic task offloading algorithm based on Lyapunov optimization to solve the problems of computational offloading and energy harvest transfer in device-to-device architectures. They verified the effectiveness of their algorithm through extensive simulation experiments, which proved that their approach reduced the average task execution time by approximately 50% compared to the baseline algorithm. Zhang and Chen [14] studied the task offloading problem in a heterogeneous MEC environment with energy harvesting and proposed a noncooperative computational offloading strategy based on game theory. Regarding MDs with limited battery and energy harvesting capabilities, Merluzzi et al. [15] designed a dynamic algorithm for jointly optimizing communication and computational resources. Kai et al. [16] investigated a collaborative computation offloading scheme and developed a collaborative computing framework in which the tasks of MDs could be partially processed at the terminals, edge servers, and cloud centers. Ren et al. [17] investigated the multiuser service latency problem in MEC offloading scenarios, proposed a new partial computation offloading model, optimized the allocation of communication and computation resources through optimal data partitioning and other strategies, and conducted experimental verification in a specific scenario where communication resources were much larger than computation resources. The proposed partial offload policy minimized the weighted delay of all user devices, thus, improving the quality of service of users. Most of the MEC offloading policy algorithms were studied mainly by assuming a priori distribution of resource demand or predicting resource demand based on historical data. In [18], the problem of offloading decisions and resource allocation among multiple users served by a base station was studied to achieve optimal system user utility, i.e., the trade-off between task delay and energy consumption. In [19], an RL-based state–action–reward–state–action (RL-SARSA) algorithm was proposed to solve the resource management problem of edge servers and to minimize the system cost (including energy consumption and computational latency) by optimizing offloading decisions. The above studies applied heuristic algorithms to deal with large-scale task offloading problems, which could take a long time to generate decisions due to the high dimensionality of the problem, and they could only find approximately optimal solutions, so they do not meet the expected requirements in practice.

In order to meet the high demand of applications for storage and computing, Ale et al. [20] studied the policy of offloading computing tasks of IoT devices to edge servers and proposed an end-to-end DRL approach to select the best edge server for offloading and allocate the optimal computational

resource such that the expected long-term utility was maximized. Chen et al. [21] proposed a distributed computation offloading method to utilize all resources, in which a complex task could be split into many small subtasks. A distributed computation offloading strategy based on a deep $Q$-learning network (DQN) was proposed to find the best offloading method to minimize the execution time of a compound task. Peng and Shen [22] described the resource allocation problem in MEC servers as a distributed optimization problem, which was solved by the MADDPG-based algorithm to maximize the number of offloaded tasks while ensuring a high quality of service.

In addition, it is difficult to develop a general and reliable task offloading strategy when applying *DRL* algorithms in a MEC environment. To address these challenges, the perceptual capabilities of deep learning, the decision-making capabilities of RL, and the fast environment learning capabilities of meta-learning can be organically combined to achieve fast and flexible optimal offloading strategies. There are many related studies on *MRL* and multiagent *RL*. Wen et al. [23] proposed a dynamic proximal policy optimization with covariance matrix adaptation evolutionary strategies based on the original proximal policy optimization to obtain a valid obstacle avoidance policy. Rakelly et al. [24] improved the sample adoption efficiency by developing an offline policy MRL algorithm that separated the task inference from the control. A meta-learning method based on expert knowledge in a sparse-reward environment was proposed in [25]. Xu et al. [26] implemented an argument-based RL approach by extracting metadata consistent with all agents, inspired by meta-learning. To address the problem that the strategy learning difficulty of exponential growth with respect to the number and type of agents, a type-based hierarchical grouping communication model was proposed in [27]. Wang et al. [28] proposed a task offloading strategy based on MRL that could quickly adapt to new environments with a small number of gradient updates. The experimental results indicated that this new offloading method could reduce the latency by 25% compared to the three baselines and could be quickly adapted to new environments.

## III. Background

### A. Reinforcement Learning

One of the best known RL algorithms is $Q$-learning [29]. This algorithm constructs mappings from state–action pairs to values with dynamic programming. These values are known as $Q$-values and are calculated with a utility function called the $Q$-function. The $Q$-function returns the expected utility of taking a given action in each state and follows a fixed policy subsequently. The collection of $Q$-values for all state–action pairs is called a $Q$-table. $Q$-learning requires several learning episodes to find an optimal $Q$-table. The problem model of the $Q$-learning algorithm is a Markov decision process (MDP) that consists of an agent, a set of states $S$, and a set of actions $A_i$ for each state $s_i \in S$ [30]. An episode is a learning period that starts from an initial state and ends when the final state is reached. During an episode, the agent chooses an action $a \in A_i$ of its current state $s$ based on its $Q$-table and the selection policy. Then, the agent perceives the next state $s'$ and receives a reward $R(s, a)$ after performing the previous action. Afterward, the agent updates its $Q$-table based on the $Q$-function

$$Q(s,a) \longleftarrow (1-\alpha)Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a' \in A} Q\left(s', a'\right)\right] \tag{1}$$

where $s \in S$, $a \in A$, $\alpha \in [0, 1]$ is the learning rate, and $\gamma \in [0, 1]$ is the discount factor. This update procedure is repeated until the agent reaches the final state, which marks the end of the episode. The main output of the $Q$-learning algorithm is a policy $\pi : S \to A$ that maximizes the sum of its rewards $R = r_0 + \gamma r_1 + \cdots + \gamma^n r_n$ for an MDP that has a terminal state $s_t$ or a termination condition. The optimal $Q$-table is given as the policy $\pi$ in $Q$-learning. The fact that $Q$-learning does not require a model of the environment is an advantage. Since $Q$-learning [29] has been proposed, RL has been widely studied, and various improved algorithms have been proposed. In [31], the double estimator was applied to a delayed $Q$-learning framework to solve the overestimation problem in RL.

### B. Meta Reinforcement Learning

MRL enhances RL methods with meta-learning, which aims to obtain a learning algorithm that can quickly determine the policy for a learning task $T_i$ drawn from a distribution of tasks $p(T)$. Each learning task $T_i$ corresponds to a different MDP, which typically shares the same state and action spaces but may differ with respect to their reward functions or dynamics. A typical example of MRL is the gradient-based MRL, which aims to learn the initial parameters $\theta$ of a policy neural network. It ensures that performing a single or few steps of policy gradient over $\theta$ with a given new task can lead to an effective policy for that task. The formulation of model-agnostic meta-learning (MAML), resulting in the target of gradient-based MRL, is shown in the following:

$$J(\theta) = \mathbb{E}_{T_i \sim p(T)}\left[J_{T_i}(F(\theta, T_i))\right] \tag{2}$$

where $J_{T_i}$ denotes the objective function of task $T_i$ and $F$ denotes the update function that depends on the objective function and optimization method. For example, if we conduct a $k$-step gradient ascent for $T_i$, then $F(\theta, T_i) = \theta + \alpha \sum_{t=1}^{k} \nabla J_{T_i}$. Therefore, the optimal meta-parameters of the policy network and update rules are

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{T_i \sim p(T)}\left[J_{T_i}(F(\theta, T_i))\right]. \tag{3}$$

The gradient-based MRL has a good generalization ability.

## IV. System Model

### A. MEC Model

The system architecture of task offloading and energy harvesting in a MEC application is demonstrated in Fig. 1. There are many different types of tasks to be processed in MD, each of which can be split into interdependent subtasks to build

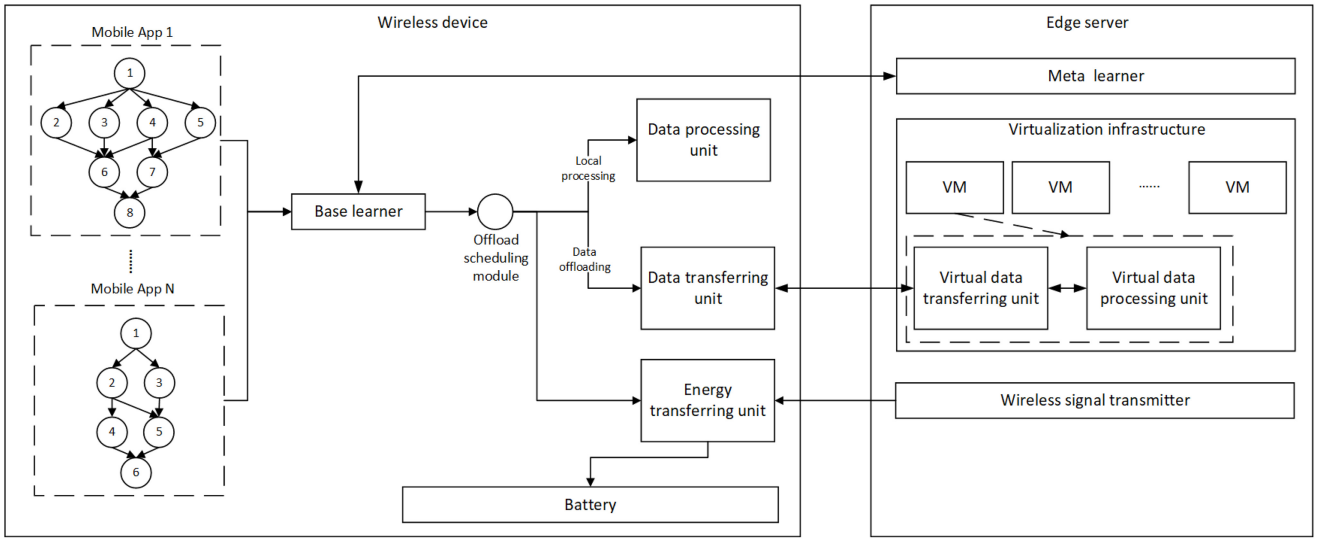



Fig. 1. System structure of task offloading and energy harvesting in MEC.

DAG dependencies. Based on the dependencies and characteristics of each DAG subtask, the offload scheduling module ranks all subtasks according to their priority, ensuring that the subtasks that depend on the previous task can be executed only after the conditions are met. Then, the offloading scheduling module generates appropriate decisions for each subtask that decide whether to offload the task and duration of the WPT service, which is executed before the offloading operation [32]. As depicted in Fig. 1, the WPT service is completed by the energy-transferring module in the MD, which receives electromagnetic waves from the edge server, converts them into electrical energy, and stores them in the device battery. The offloading decision also determines how the task is executed, that is, the DAG subtask can be processed directly on the device or sent to the edge server for processing via the data transferring unit.

A learner is included in each MD and edge servers, where the base learner is trained in the MD. Here, the base learner is responsible for generating a specific policy based on task features and requires less iterations. In contrast, the meta-learner is trained at the edge server, which needs to train and acquire meta-policies based on different tasks; hence, it is relatively abstract in the learning space compared to the base learner, which is independent of the specific task. Based on this design characteristic, the base learner first acquires and updates the meta-policy parameters. Subsequently, a specific offloading strategy is trained and generated based on the meta-policy and task features, which are uploaded to the meta-learner in the form of higher order metadata. The strategy ensures that all specific policies can obtain the same structured state. Finally, the meta-learner trains and updates the parameters based on the metadata and waits for the base learner to be called again, starting a new cycle. Because the meta-learner requires more training samples and its learning cost is too high, deploying it on the edge server helps reduce the training time to improve the learning efficiency of new tasks. On the other hand, the base learner performs fast iterations based on the meta-policy and consumes less resources and power in the MD.

### B. Problem Model

For the task offloading problem in MEC with multiple DAG types, the DAG application is denoted as $G = (V, E)$, where $v_i \in V$ is a subtask in the DAG application, $e(v_j, v_i) \in E$ indicates that subtask $v_i$ depends on subtask $v_j$, $v_i$ is a posterior subtask of $v_j$, and $v_j$ is a preceding subtask of $v_i$. A posterior subtask can be executed only when all its preceding subtasks are completed. For each subtask $v_i$, we set its input data volume $D_i^{\text{in}}$, CPU computation cycle number $C_i$, output data volume $D_i^{\text{out}}$, and maximum execution time $T_i^{\max}$ accordingly. For the MD, the total CPU clock frequency and battery capacity are set to $F^l$ and $B^{\max}$, respectively. For the edge server, the upload and download rates between the edge server and MD are set to $R^{\text{up}}$ and $R^{dw}$, respectively, while its CPU clock frequency is set to $F^r$. In addition, considering that the computational and network resources of the MD are limited, all processing units in MDs and edge servers are configured with respect to corresponding task queues [33]. When multiple DAG subtasks compete for device resources, they must be computed or transmitted according to their priority [34].

Offloading strategy $A = \{a_1, a_2, \ldots, a_n\}$ is a sequence set consisting of individual strategies for each subtask, where each strategy contains the task scheduling type (local execution or offloading execution) and WPT service duration $CT_i$. In this study, to illustrate the dependency relationship between DAG subtasks and their processing order in the execution process, for subtask $v_i \in V$, we set its task sending completion time as $FT_i^{\text{up}}$, remote execution completion time as $FT_i^r$, data return completion time as $FT_i^{dw}$, and local execution completion time as $FT_i^l$ accordingly. When the task scheduling type is offloading execution, it follows that $FT_i^{dw} > FT_i^r > FT_i^{\text{up}} > CT_i$. When the task scheduling type is local execution, it follows that $FT_i^l > CT_i$; that is, the WPT service is executed first to prevent the task from failing owing to insufficient power when data are uploaded or processed. The posterior subtask $v_i$ that depends on $v_j$ can request resources to process the relevant data only when its timestamp is greater than $FT_j^{dw}$

or $FT_j^l$ according to the scheduling type. This ensures that all preceding business data of this posterior subtask are complete. To reflect the impact of resource competition among multiple DAG subtasks, the transmitting resource availability time $AT_i^{\text{up}}$, remote execution availability time $AT_i^r$, data return availability time $AT_i^{dw}$, and local execution availability time $AT_i^l$ are set accordingly. Therefore, based on the relevant parameters of the MD and the edge server, the latency and the energy consumption generated by task $v_i$ for performing local scheduling and offloading scheduling can be calculated.

## V. Algorithm Design

This study proposes an offloading strategy combining DRL and meta-learning. Therein, DRL is a sequential decision-making method that learns the target environment through continuous trial and error, which is beneficial for solving complex problems, such as task offloading in the MEC. Meta-learning is a method of learning multitask generalization capability, which first generalizes a model by learning multiple task-specific models. Then, it learns a specific task based on a small number of samples. Therefore, by combining meta-learning with DRL, prior knowledge from a large number of RL tasks can be obtained to improve learning and convergence abilities when facing new tasks. Moreover, this approach can increase the algorithm flexibility and facilitate the lowering of learning costs to quickly solve the offloading problem of different DAG task structures in a MEC environment. In this study, we solve the edge server decision making and WPT service duration problems based on the multiagent MRL algorithm.

### A. MDP Model

To solve the offloading problem with multiple DAG tasks using MRL, this study models the offloading process of a DAG task as an MDP in the following form: $M = (S, A, P, P_0, R, \gamma)$. Therein, $M$ denotes the offloading model of each subtask, $S$ is its corresponding state space, $A$ is the action space, $P$ is the state transfer matrix, $P_0$ is the initialized state distribution, $R$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. For each subtask $v_i$, the agent observes the current MEC environment as $s_i \in S$. Subsequently, it selects an action $a_i \in A$ from the action space based on its learned offloading strategy $\pi(a_i|s_i)$, which is executed in the environment to obtain a reward value $r_i \in R$ and a new state $s_{i+1} \in S$. This process is repeated to construct a sequence set $A_n = (a_1, a_2, \ldots, a_n)$ of the offloading decisions for each subtask. Furthermore, to apply MRL to the MDP model, the MDP learning process is decomposed into two parts: learning a meta policy efficiently across all MDPs and learning a specific offloading strategy for an MDP quickly based on the learned meta policy. To solve the MEC offloading problem for multiple DAG tasks, detailed definitions of the state space, action space, and reward function utilized in the MDP are given in the following.

*1) State Space:* Because the energy consumption and latency required for the DAG task offloading are closely related to the task type, task data volume, and state of the MEC environment, the state space is denoted as $s_i = (G, A_i, D_i^in, C_i, D_i^{\text{out}}, T_i^{\max}, B_i)$ for subtask $v_i$. $G$ denotes the topological ordering of the currently scheduled DAG tasks, which is aligned with the task priority order. Each subtask contains its detailed information, such as task index, task computation volume, upload data volume, download data volume, and maximum allowed time. $A_i = (a_1, a_2, \ldots, a_{i-1})$ denotes the offloading decisions made by all preceding subtasks whose priorities are higher than that of subtask $v_i$. $D_i^{\text{in}}$ is the input data volume for subtask $v_i$, $C_i$ is the number of CPU computation cycles it requires, $D_i^{\text{out}}$ is the output data volume, $T_i^{\max}$ is the maximum execution time allowed for the subtask, and $B_i$ is the remaining electric energy of the MD before executing the offloading scheduling. Furthermore, to alleviate the impact of the disparity in the number of subtasks in different DAG tasks on the state space, the topological ordering $G$ is set to contain two vectors for storing the front and back task indices; the length of the two vectors is kept as the same, and the position of the insufficient length is set to $-1$.

*2) Action Space:* To address the impact of the WPT service duration on latency and task success rate, this study determines the task offloading decision and the WPT service duration by designing two agents with the same state space but different action spaces. The discrete action space is specified as $a^{\text{offload}} = \{0, 1\}$ in the task offloading decision problem, with 0 indicating that the subtask is executed on the MD and 1 indicating that the subtask is offloaded to the edge server for execution. Regarding the WPT service duration problem, the continuous action space is specified as $a^{\text{wpt}} = \lambda$. $\lambda$ denotes the duration, whose accuracy is retained to two decimal places. This value directly affects the latency and the remaining electric energy of the subtask; thus, indirectly affects the scheduling decision of the subsequent subtasks.

*3) Reward Function:* The multiagent offloading strategy based on the task scheduling decision and the WPT service duration is a fully cooperative game process, where the optimization goal of each agent is to minimize the energy consumption and the latency, as well as to improve the task success rate. The energy consumption and the latency of the $i$th subtask in the task priority queue are obtained by taking the difference of the corresponding values belonging to two neighboring subtasks, which is formulated as

$$\begin{cases} \Delta E_i = E_i^{\text{total}} - E_{i-1}^{\text{total}} \\ \Delta T_i = T_i^{\text{total}} - T_{i-1}^{\text{total}} \end{cases} \tag{4}$$

In addition, when the electric energy of the MD is insufficient or the latency of the subtask exceeds the maximum allowable time, a failure processing triggers, which directly affects the user service experience. Therefore, it is necessary to avoid such a situation. The formula for the failure processing is as follows:

$$I_i(\text{Fail}) = 1 \quad \text{if } \Delta E_i > B_i \text{ or } \Delta T_i > T_i^{\max} \tag{5}$$

where $B_i$ denotes the remaining electric energy after processing the $i$th subtask and $T_i^{\max}$ denotes the maximum processing time allowed for the $i$th subtask. Based on the above analysis, the reward function for the offloading decision of the $i$th

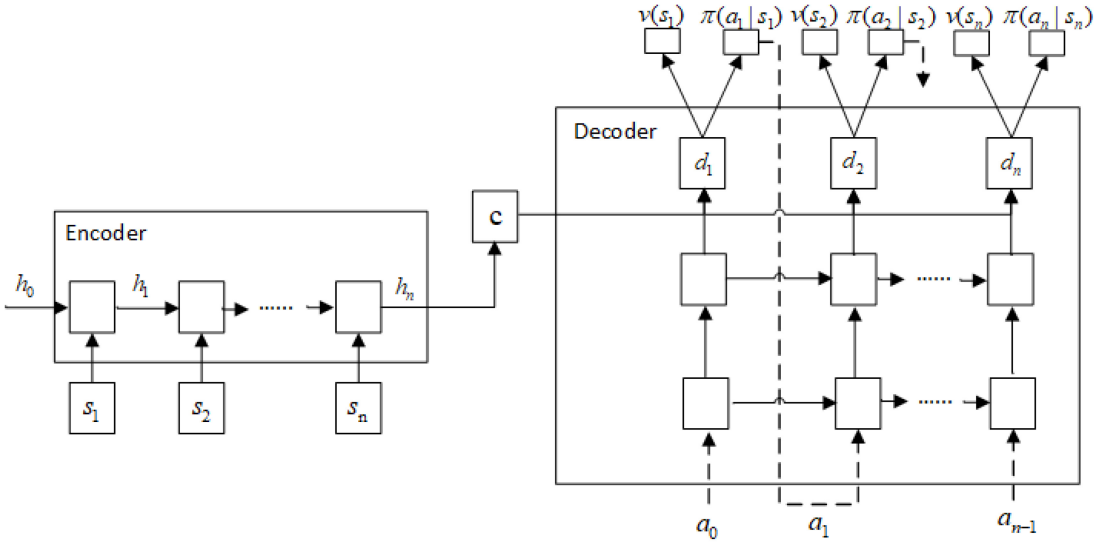



Fig. 2. Structure of the Seq2Seq network.

subtask is

$$\begin{cases} r_i = -((\sigma \Delta E_i + \tau \Delta T_i) \cdot (1 - I_i(\text{Fail})) + \omega \cdot I_i(\text{Fail})) \\ \sigma + \tau = 1 \end{cases} \tag{6}$$

where $\sigma$ and $\tau$ denote the weights of energy consumption and latency, respectively, and $\omega$ is the penalty value for the subtask's failure processing.

### B. Multiagent Meta Reinforcement Learning Algorithm

Considering the structure of the DAG task, this study uses a Seq2Seq network model to represent the scheduling process of subtasks in order of the increasing priority, to realize the impact of task scheduling priority on the offloading strategy. To solve the task offloading problem for mobile applications with multiple DAG types and edge servers with WPT services, this study proposes a multiagent RL algorithm-based policy, namely, MASAC, to generate the corresponding offloading policy. MASAC contains two agents that cooperate with each other and share the same optimization goal of minimizing the energy consumption and the latency. The policy takes whether to offload the task and the WPT service duration as the respective learning objectives for the two agents. Such an approach can reduce the energy consumption and the latency required for task processing, and increase the task success rate as much as possible. To improve the generalization of the policy, this study combines MASAC with the idea of meta-learning, proposes a multiagent MRL-based method, dubbed meta-MASAC. It first learns each task offloading policy to generate a meta-policy, and then combines new task features to generate a specific offloading policy after a short training interval. The implementation of each algorithm is described as follows.

*1) Seq2Seq:* Because the offloading strategy of the DAG task comprises a series of offloading strategies for all subtasks and for the state space $s_i$ of subtask $v_i$, its DAG task structure is different, which further leads to differences in the length of its topological ordering $G$. Therefore, to ensure that the subtask priority order is consistent with the generated decision sequence and the difficulties of network training caused by inconsistent state dimensions are addressed, this study uses a Seq2Seq network to represent the process of generating offloading strategies for subtasks in increasing order of priority. The network encoder compresses the input sequence into a uniform vector context, and then, the network decoder decodes and generates the specified sequence according to the context [35]. The decoding process of the network decoder continuously takes the output of the previous moment $(t-1)$ as the input of the next moment $t$ and decodes it cyclically until the output is the stop sign. Note that the Seq2Seq network can adjust its time length according to the number of subtasks and does not need to ensure that the length of the input or output sequence remains the same for each DAG task. Therefore, it can be used to handle DAG tasks with different structures.

The Seq2Seq network structure combining subtask priorities is presented in Fig. 2, where both the encoder and decoder are composed of the long short-term memory (LSTM) network. The input of the encoder is a sequence of subtasks, while the output of the decoder is the offloading strategy corresponding to each subtask. In addition, to avoid the accuracy degradation caused by a long vector context, this study uses the attention mechanism to assign weights to each output of the vector context and decoder. Now, suppose that the input sequence of states by the encoder is Input $= (s_1, s_2, \ldots, s_n)$ and the corresponding sequence of actions output by the decoder is Output $= (a_1, a_1, \ldots, a_n)$. Then, the hidden state $h_i$ of the $i$th encoder is as follows:

$$h_i = f_{enc}(s_i, h_{i-1}) \tag{7}$$

where $f_{enc}$ denotes the encoding function. After encoding all subtasks, the output $d_j$ of the decoder at the current moment must be calculated based on the vector context and the output $d_{j-1}$ of the decoder at the previous moment, which is formulated as

$$d_j = f_{dec}\big(c_j, d_{j-1}, a_{j-1}\big). \tag{8}$$

The corresponding formula for the vector context at the $j$th output is stated as follows:

$$c_j = \sum_{i=1}^{n} \mu_{ji} h_i \tag{9}$$

where $\mu_{ji}$ is the weight of the hidden layer state $h_i$ in the encoder, which can be calculated with the SoftMax function. It is formulated as

$$\mu_{ji} = \text{softmax}\big(e\big(h_i, d_{j-1}\big)\big). \tag{10}$$

Therein, function $e$ is the distance calculation function, which denotes the effect of the hidden state $h_i$ on the decoder's output $d_{j-1}$. To combine Seq2Seq networks with DRL algorithms, the output of Seq2Seq networks can be input into a fully connected network to approximate the $Q$-value function $v(s_i)$ and the policy function $\pi(a_i|s_i)$ for training the actor and critic networks.

*2) MASAC:* To address the problem that agents need to decide the offloading strategy and the WPT service duration simultaneously, this study utilizes the method proposed in [36]. This method uses multiple agents to solve the problem of an unstable environment and difficult convergence by combining the MADDPG and SAC algorithms, where the MADDPG algorithm optimizes each agent's policy as a whole via centralized training and distributed execution to reduce the algorithm variance [22]. The SAC algorithm introduces maximum entropy in the reward function to ensure that as many action possibilities as possible are explored, thereby enhancing the exploration ability and robustness. Thus, the MASAC algorithm based on MADDPG and SAC follows the core ideas of centralized training and decentralized execution. Moreover, it explores as many optimal paths as possible, which helps reduce the energy consumption and the latency of DAG subtasks. Note that the MASAC algorithm is used to solve the offloading strategy problem for different subtasks in the same DAG task. Each subtask contains two agents that cooperate with each other and share the same optimization goal of minimizing energy consumption and latency. They only differ regarding the actions taken. One agent is used to select the appropriate offloading strategy based on the state, that is, whether the current subtask is offloaded to the edge server for execution. Another agent is used to calculate the WPT service duration to avoid the failure of subtask processing owing to insufficient power. Now, assume that the parameters of the strategy network $\pi_i(s;\theta_i)$ and the evaluation network $Q_i(x, a_1, a_2, \ldots, a_N; \varphi_i)$ for each agent in the MASAC algorithm are $\theta_i$ and $\varphi_i$, respectively. Consequently, the action and expected reward for the $i$th agent are given as

$$\begin{cases} a_i = \pi_i(s;\theta_i) \\ y_i = r_i + \gamma\left(Q_i'\big(x, a_1', a_2', \ldots, a_N'; \varphi_i'\big) + \alpha \sum_{i}^{N} H_i\big(\pi_{\theta_i}\big)\right) \end{cases} \tag{11}$$

where $r$: $S \times A \rightarrow R$ denotes the reward function, $s \in S$ denotes the current state, $a \in A$ denotes the action taken, $x = (s_1, s_2, \ldots, s_N)$ denotes the state set of all agents, $\varphi'$ denotes the target evaluation network parameter, $a_i' = \pi_i'(s)$ denotes the action selected by the $i$th agent based on the target policy network, $\gamma \in [0, 1]$ denotes the discount factor, $\alpha > 0$ is used to control the weight factor of entropy, and $H(\pi_\theta) = -\log(\pi_\theta)$ denotes the entropy of strategy $\pi$ in state $s$. Thus, each agent can determine the TD error based on the current policy of the actor network and estimate the value of the critic network, which is calculated as

$$\Lambda^{\pi_{\theta_i}}(x, a) = y_i - Q_i(x, a_1, a_2, \ldots, a_N; \varphi_i). \tag{12}$$

Then, the critic network calculates the loss function, which is given as

$$L = \frac{1}{N} \sum_{i=1}^{N} \big(\Lambda^{\pi_{\theta_i}}(x, a)\big). \tag{13}$$

The updated network parameters are subsequently calculated with the gradient descent method, depicted as

$$\nabla_\theta J = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_i} \pi_i(s;\theta_i) \nabla_{a_i} Q_i(x, a_1, \ldots, a_N). \tag{14}$$

*3) Meta MASAC:* To ensure that the offloading strategy is applicable to many different structures of DAG tasks, this study combines the MASAC algorithm and meta-learning to enable learning the meta-information in the subtask offloading strategies for initializing the parameters of the actor and critic networks. Subsequently, the parameters are adapted with a small amount of training using specific task data, so that the model can be used as an offloading strategy for new subtasks after self-adaptation. The meta-learning algorithm is composed of two parts: 1) a base learner and 2) a meta-learner. The base learner is responsible for inner loop training in the task space and provides feedback to the meta-learner in a higher order form. The meta-learner is used for outer loop training in an abstract space unrelated to a specific task. It can obtain metadata from different tasks and use them to update the parameters of the base learner and meta-learner to achieve fast learning and adaptation [37]. As depicted in Fig. 3, the meta-MASAC algorithm can be divided into the following four steps.

*a) Base learner training:* The MASAC algorithm is used in this step to train the $N$ agents centrally and update the actor network parameters separately. The generated set of corresponding policies is $\pi = (\pi^1, \pi^2, \ldots, \pi^N)$, where $\pi^i$ denotes the policy network of the $i$th agent.

*b) Meta information storage:* $N$ agents perform actions at $T$ time steps according to the policy. Their corresponding observed states $o$, actions $a$, and the set of $Q$-values are $Exp = \{(o^1, a^1, q^1)_{(1,2,\ldots,T)}, (o^2, a^2, q^2)_{(1,2,\ldots,T)}, \ldots, (o^N, a^N, q^N)_{(1,2,\ldots,T)}\}$, respectively, where $(o^i, a^i, q^i)_t$ denotes the feedback data obtained by the $i$th agent at time step $t$. The corresponding data of all agents at $T$ time steps is stored centrally to ensure that they can be used to learn the metadata of all tasks.

*c) Meta-learner training:* Data collected by all agents at $T$ time steps is uploaded to the meta-learner. Subsequently, the meta-actor and meta-critic use this data to train the policy network actor and the evaluation network critic, respectively. Regarding the policy network actor, its training data is the set of state–action pairs of all agents $(O, A) =$

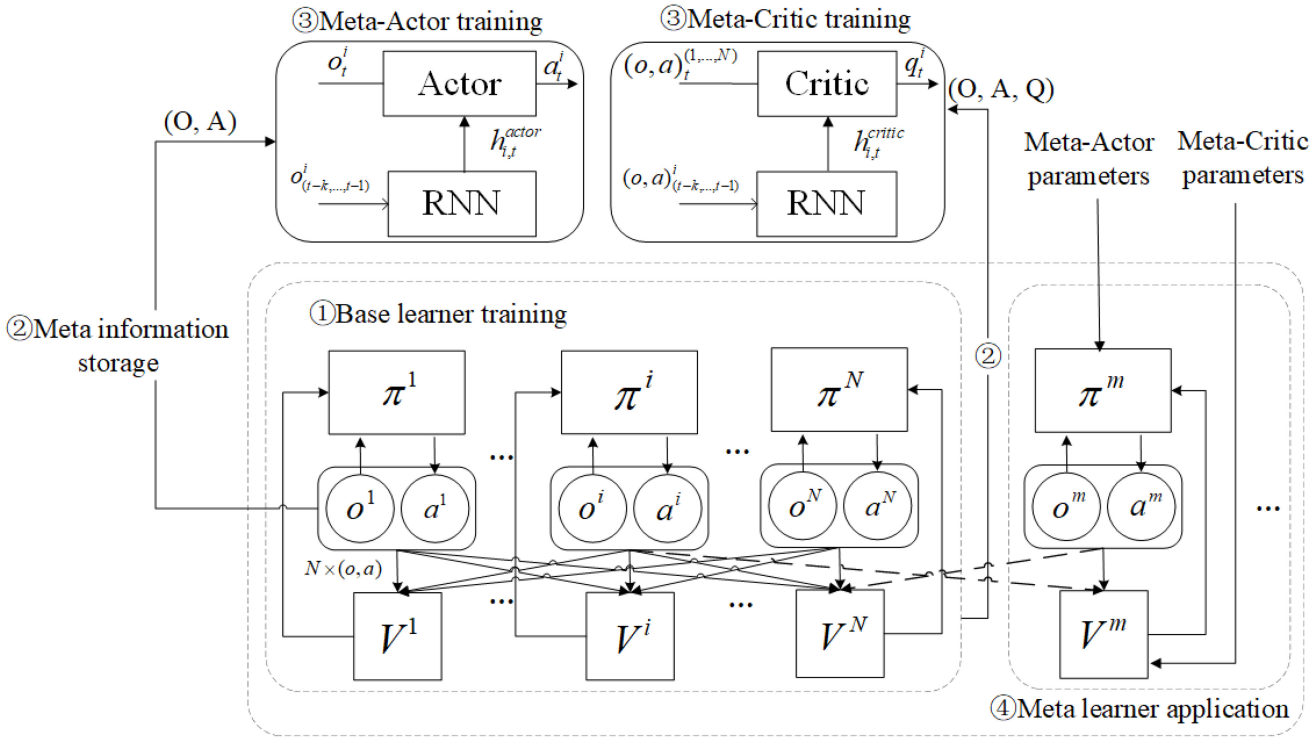



Fig. 3. Processing of Meta MASAC algorithm.

$\{(o^1, a^1)_{(1,2,\ldots,T)}, \ldots, (o^N, a^N)_{(1,2,\ldots,T)}\}$. Because the goal of the policy network actor is to obtain the maximum reward, the corresponding parameter updating formula is

$$\theta_i = \arg\max_{\theta_i} \; y_i((x, a)_t) \tag{15}$$

where $y_i((x, a)_t)$ is the expected reward of the $i$th agent, which can be calculated using (29), $x = (o_1, o_2, \ldots, o_N)$ denotes the set of observed states of all agents, and $\theta_i$ denotes the policy network parameter. The evaluation network critic evaluates the actor state and action using the training data $((O, A), Q) = \{(o^1, a^1)_{(1,2,\ldots,T)}, \ldots, (o^N, a^N)_{(1,2,\ldots,T)}, q^1_{(1,2,\ldots,T)}, \ldots, q^N_{(1,2,\ldots,T)}\}$. The training of the critic network is based on this data to estimate the expected reward as accurately as possible. Therefore, the parameter updating formula of the evaluation network critic is

$$\varphi_i = \arg\min_{\varphi_i} \Lambda^{\pi_{\theta_i}}((x, a)_t) \tag{16}$$

where $\Lambda^{\pi_{\theta_i}}((x, a)_t)$ is the loss function of the $i$th agent, which can be calculated using (30) and $\varphi_i$ is the evaluation network's parameter. In addition, because the state information of the MEC environment is partially observable, the LSTM network is needed in both the meta-actor and meta-critic to analyze the data within the last $k$ time steps to obtain more accurate state information through the connection among the data. One reason without the entire historical experience is that a smaller data volume can improve the computational efficiency. Another reason is that the latest experience has more influence on the current policy. In addition, because the meta-actor only needs the historically observed states $o_t$ as training data, the calculation formula becomes

$$h^{\text{actor}}_{i,t} = \text{LSTM}_{\omega}\left(o^i_{(t-k,\ldots,t-1)}\right) \tag{17}$$

where $h^{\text{actor}}_{i,t}$ denotes the output value of the LSTM network of the meta-actor in the $i$th agent at the current time step, $\omega$ denotes the parameters of the LSTM network in the meta-actor, and $o^i_{(t-k,\ldots,t-1)}$ denotes the observed state of the $i$th agent at previous $k$ time steps. The meta-critic aims to compute a reasonable $Q$-value based on the state–action pairs of the agent. Therefore, its LSTM network requires the observed state $o_t$ and action $a_t$ as training data, which is calculated as follows:

$$h^{\text{critic}}_{i,t} = \text{LSTM}_{\mu}\left((o, a)^i_{(t-k,\ldots,t-1)}\right) \tag{18}$$

where $h^{\text{critic}}_{i,t}$ denotes the LSTM network's output value of meta-critic in the $i$th agent, $\mu$ denotes the parameters of the LSTM network in meta-critic, and $(o, a)^i_{(t-k,\ldots,t-1)}$ denotes the state–action pairs of the $i$th agent in the previous $k$ time steps.

Both the meta-actor and meta-critic need to update their network parameters based on metadata. Now, suppose that the parameters of the corresponding output layer are $\tau$ and $\delta$, respectively. The corresponding parameter updating formulas are defined as follows:

$$\omega, \tau \leftarrow \arg\min_{\omega,\tau}\left(a^i_t - \pi^{\text{mac}}_{\tau}\left(o^i_t, h^{\text{actor}}_{i,t}\right)\right), i \in [1, 2, \ldots, N] \tag{19}$$

$$\mu, \delta \leftarrow \arg\min_{\mu,\delta}\left(q^i_t - v^{\text{mcr}}_{\delta}\left((o, a)^{(1,2,\ldots,N)}_t, h^{\text{critic}}_{i,t}\right)\right) \quad i \in [1, 2, \ldots, N] \tag{20}$$

where $\pi^{\text{mac}}_{\tau}(o^i_t, h^{\text{actor}}_{i,t})$ is the policy function of the meta-actor, which outputs the corresponding action when the

**Algorithm 1** Training Meta-Learning Algorithm

**Input:** Trained actor and trained critic of MASAC with $N$ agents;
**Output:** Trained models of meta-actor and meta-critic.

Initialize output parameter $\tau$ and LSTM parameter $\omega$ of meta-actor, output parameter $\delta$ and LSTM parameter $\mu$ of meta-critic, experience replay buffer $D$.
**for** $step = 1 \rightarrow T$ **do**
  Every agent executes action which generate by $\pi = \{\pi^1, \pi^2, \ldots, \pi^N\}$;
  Add $\{(o^1, a^1, q^1)_{step}, (o^2, a^2, q^2)_{step}, \ldots, (o^N, a^N, q^N)_{step}\}$ to $D$;
**end for**
**for** agent $i = 1 \rightarrow N$ **do**
  **for** each $(o^i, a^i)_{step}$ **do**
    Generate temporal mini-batch $o^i_{(t-k,\ldots,t-1)}$;
    Train output network $\pi^{mac}_{\tau}(o^i_t, h^{actor}_{i,t})$ and LSTM network $h^{actor}_{i,t}$ of meta-actor;
    Calculate error and update parameter of meta-actor according to Eq. (19);
  **end for**
  **for** each $(o^i, a^i, q^i)_{step}$ **do**
    Generate temporal mini-batch $(o, a)^i_{(t-k,\ldots,t-1)}$;
    Train output network $V^{mcr}_{\delta}((o, a)^{(1,2,\ldots,N)}_t, h^{critic}_{i,t})$ and LSTM network $h^{critic}_{i,t}$ of meta-critic;
    Calculate error and update parameter of meta-critic according to Eq. (20);
  **end for**
**end for**
**return** Meta-actor and meta-critic

current observed state and the historical state based on LSTM are known and updates the parameters by reducing the difference between the current and the actual actions. $V^{\text{mcr}}_{\delta}((o, a)^{(1,2,\ldots,N)}_t, h^{\text{critic}}_{i,t})$ represents the evaluation function of the meta-critic, which can calculate the corresponding $Q$-value by inputting the state–action pairs at the current moment and the historical information output by LSTM. Then, the $Q$-value is compared with $q^i_t$ in the data to update the network parameters. Based on the above analysis, the algorithm flow of the meta-learner training is presented in Algorithm 1.

*d) Meta-learner application:* The result of training meta-learners, namely, meta-actor and meta-critic, can be used to initialize base learners, such as the actor network and the critic network. Subsequently, a short training is performed on the generic model according to the specific task characteristics to fine-tune the network parameters and generate a specific model based on this task. The application flow is reported in Algorithm 2, where meta-actor is a meta-strategy network that outputs actions directly based on states, and meta-critic is a value network that outputs values based on states. During training, both networks are optimized synchronously, and only the meta-actor network is used to output actions during testing.

**Algorithm 2** Applying Meta-Learning Algorithm

**Input:** Trained meta-actor and meta-critic model, the number $N$ of new tasks, iteration times $T$;
**Output:** Action of new task.

**for** $j = 1 \rightarrow N$ **do**
  Initialize actor network parameter $\theta_j$ Of agent $j$ based on meta-actor;
  Initialize critic network parameter $\varphi_j$ Of agent $j$ based on meta-critic;
**end for**
**for** $step = 1 \rightarrow T$ **do**
  **for** $j = 1 \rightarrow N$ **do**
    Update parameter $\theta_j$ and $\varphi_j$ according to Eqs. (15) and (16);
  **end for**
**end for**
**for** $j = 1 \rightarrow N$ **do**
  Take action according to Eq. (11);
**end for**
**return** the action

## VI. Experiments and Analysis

### A. Simulation Environment Settings

The simulation experiments are to be conducted in the TensorFlow framework on the offloading problem of different DAG tasks in a MEC environment. We use the EUA data set [38], which contains computational performance, remaining power, charging power, location information, and computation of different application services for MDs in multiple regions. Subsequently, the advantages and disadvantages of offloading strategies in terms of latency, energy consumption, and task success rate are demonstrated experimentally. The compared algorithms include the strategy LE based on local execution, the strategy OE based on offloading execution, the strategy based on the HEFT algorithm, the strategy based on the multiagent DRL algorithm MADDPG, and the strategy based on multiagent MRL algorithm Meta MASAC. To test whether the MRL algorithms have excellent learning and convergence capabilities when faced with new tasks, numerous DAG task structures must be constructed for training tests. The configuration of some basic experimental parameters is referred to the excellent

work done in [28]. Moreover, it is known that the DAG task structures are mainly influenced by the following parameters according to [39].

1) *Total Number of Subtasks:* It is used to set the number of subtasks in the entire DAG task. The larger this value is, the higher the DAG task scale is, which increases the corresponding decision sequence length. The total number of subtasks is set to $n = 20$ in the experiments by default.
2) *DAG Width:* It is used to control the height and width of the entire DAG task structure. When the total number of subtasks remains the same, the larger the width value is, the wider the DAG task structure is, and vice versa. This value is randomly taken from the set $\{0.2, 0.4, 0.6, 0.8\}$ when generating DAG tasks.
3) *DAG Density:* It is used to control the degree of dependence of each subtask between adjacent levels in the DAG task. The larger this value is, the more dependent the subtasks are to their predecessors. This value is taken randomly from the set $\{0.5, 0.6, 0.7, 0.8\}$ when generating a DAG task.
4) *Computation–Communication Ratio:* It is used to set the ratio between computation and communication costs in DAG tasks. A larger value indicates that the task prefers high computational demands and vice versa. Because most subtasks are computationally intensive, the value is in the range of $[0.5, 0.7]$.

Now, assume that the input data volume $D_i^{\text{in}}$ and output data volume $D_i^{\text{out}}$ of subtask $i$ are in the range of $[10, 100]$ kB, the required CPU cycles $C_i$ of subtask are in the range of $[10^7, 10^8]$, and the maximum allowed execution time $T_i^{\max}$ of the subtask is in the range of $[500, 1500]$ ms. The CPU clock frequencies of MD and the edge server in such a MEC environment are set to 1.5 and 2.0 GHz, respectively. The corresponding upload and download rates are $R^{\text{up}} = R^{dw} = 7$ Mb/s, the upload and download power levels are $P^{\text{up}} = 0.5$ W and $P^{dw} = 0.6$ W, respectively, and the battery capacity of the MD is 18.5 Wh. In addition, the parameters of Meta-MASAC need to be set, which are listed in detail in Table I.

TABLE I
PARAMETER SETTING OF ALGORITHMS

| Parameters | Value |
|---|---|
| Memory space size $M$ | $1 \times 10^6$ |
| Maximum number of training iterations | 1000 |
| Batch learning size $K$ | 32 |
| Optimizer | Adam |
| Target network update period | 100 |
| Discount factor | 0.99 |
| Encoder network type | LSTM |
| Number of hidden layers of Encoder network | 2 |
| Number of hidden cells of Encoder network | {256,256} |
| Decoder network type | LSTM |
| Number of hidden layers of Decoder network | 2 |
| Number of hidden cells of Decoder network | {256,256} |
| Number of hidden layers in the Actor network | 2 |
| Number of hidden cells in the Actor network | {500,128} |
| Actor network activation function | ReLU |
| Actor learning rate | $1 \times 10^{-3}$ |
| Number of hidden layers of Critic network | 3 |
| Number of hidden units in Critic network | {1024,512,300} |
| Critic network activation function | ReLU |
| Critic learning rate | $1 \times 10^{-3}$ |
| Index storage length of pre-task and post-task | 12 |
| Number of base learner iterations | 10 |

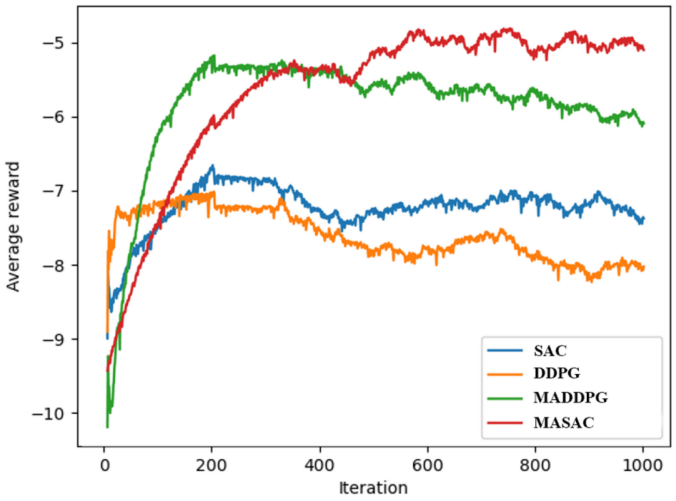



Fig. 4. Average reward by RL algorithms.

## B. Results and Analysis

To verify the validity of MASAC in the task offloading problem, we trained the MASAC offloading strategy using a specific set of DAG tasks. The set contains 100 different DAG tasks sharing the same parameters: total number of subtasks = 20; DAG width = 0.4; DAG density = 0.5; and computation–communication ratio = 0.5. Finally, the performance of the algorithm is verified using a test set with the same parameters and different structures. Fig. 4 demonstrates the results of the average reward value obtained by each DRL algorithm during the training process. The larger the value is, the better the result of the offloading policy is. Observe that the DDPG and the SAC algorithms have worse convergence results in a multi-intelligence environment, where the SAC algorithm achieves a higher reward value after convergence but a slower convergence speed compared to the DDPG algorithm. This is mainly because the SAC algorithm requires more iterations to explore more decision paths, which leads to obtaining better solutions. In addition, MADDPG and the improved MASAC algorithm perform better in a multi-intelligent environment, and the proposed algorithm obtains a higher reward value after convergence.

After training, we selected four DAG tasks with the same parameters as those of the training set but with different structures to test the RL algorithms. As depicted in Fig. 5, the policies generated by SAC and DDPG exhibit average performance in all metrics, which is mainly owing to the unstable training results of these two algorithms in a multi-intelligence environment and struggling when converging to the optimal solution. In contrast, the MADDPG algorithm can effectively learn stable policies using centralized training and distributed execution. Furthermore, its energy consumption, latency, and task success rate are better than those of DDPG and SAC. The improved MASAC algorithm has the

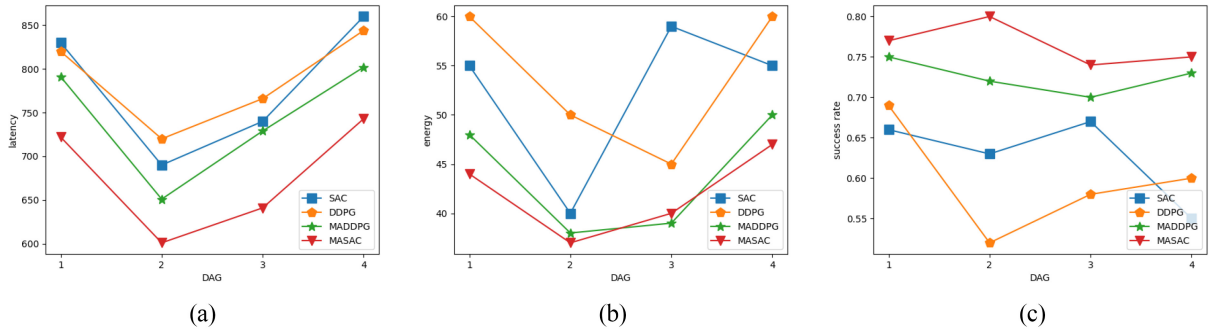



Fig. 5. Comparison of each trained RL algorithms in specific DAG tasks. (a) Latency comparison. (b) Energy comparison. (c) Success rate comparison.

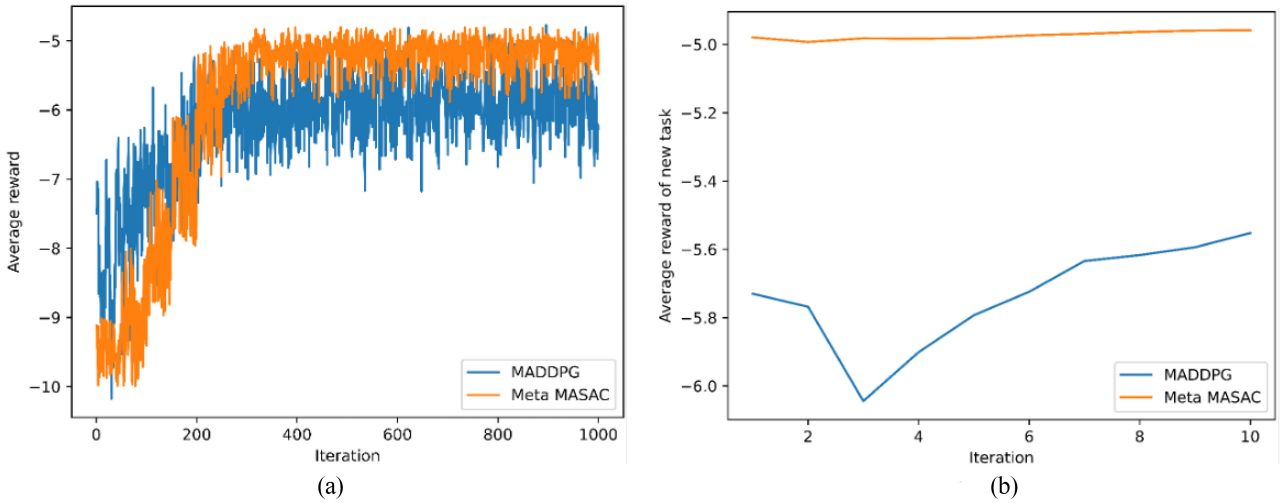



Fig. 6. Meta MASAC average reward during training. (a) Average reward during training. (b) Average reward of new task training.

best performance among DRL algorithms with respect to all metrics.

To reflect the convergence performance of the algorithms in the task offloading problem, we trained the DRL-based offloading strategy in a default environment with 20 subtasks. As depicted in Fig. 6(a), the average reward values of MADDPG and meta-MASAC during the training process are close. It is observed that both algorithms can effectively converge after sufficient training. Compared to the MADDPG algorithm, the meta-MASAC algorithm achieves a relatively better final convergence value, although its early convergence speed is slower. This is mainly because the meta-MASAC algorithm can find a better solution by exploring more actions, but at the expense of more search time. In addition, to assess the offloading performance of the algorithm in new tasks, after the training process in the default environment, a new DAG task structure with the same number of subtasks is fed into the current trained network to obtain the corresponding offloading strategy.

Fig. 6(b) shows the average reward value when MADDPG and meta-MASAC are applied to a new task structure with a small number of iterations. The figure demonstrates that the meta-MASAC algorithm converges faster and can be quickly applied to solve new tasks, whereas the MADDPG algorithm is less effective when faced with new tasks. Thus, it can only be applied to specific tasks that it has already been trained on.

To further illustrate the performance of each offloading strategy when applied to a new DAG task structure, a small number of iterations are performed on each algorithm to reflect its performance with respect to the convergence effect in terms of energy consumption, latency, and task success rate. Fig. 7 shows that the local execution strategy LE achieves the worst iterative results in terms of energy consumption and latency. Moreover, its task success rate cannot be guaranteed because of the limited performance and power of MDs. Therefore, the task's success rate is not recorded to avoid confusion. The offloading execution strategy OE exhibits the best performance in terms of energy consumption but a poor performance in terms of latency and task success rate. The main reason is that the energy consumption generated by transmission in MDs is much smaller than that generated by computation. When all subtasks are offloaded to the edge servers for processing, the total energy consumption is only related to the transmission energy consumption. However, a large amount of data transmission occupies a large network bandwidth, increasing its transmission latency. Thus, the poor performance of the OE strategy in terms of latency further affects its task success rate. The HEFT-based strategy performs better in terms of latency and has a balanced performance in terms of energy consumption and task success rate. This is mainly because the HEFT algorithm ranks the tasks based on weights, then schedules them according to the earliest completion time, fully leveraging the computational resources of MDs and edge servers. Therefore, the task completion latency is guaranteed. The MADDPG-based strategy can converge in terms of energy

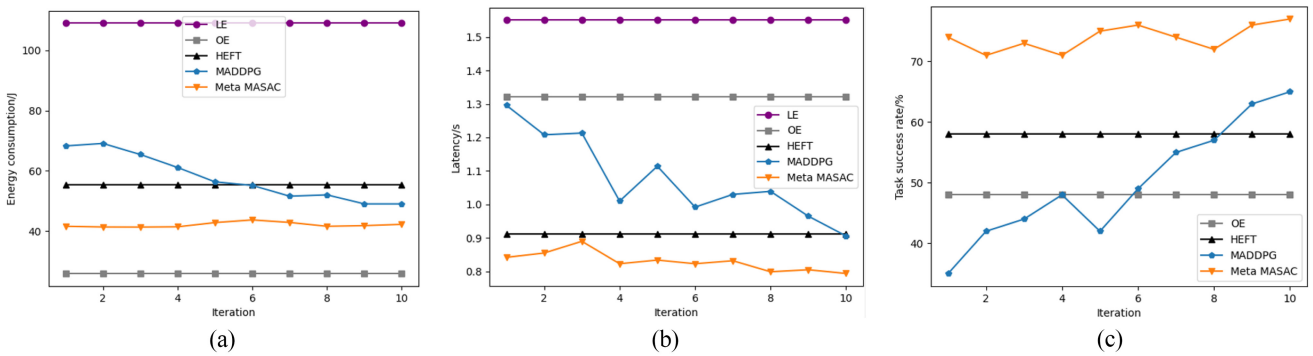



Fig. 7. Comparison of each strategy with respect to different metrics after a few iterations. (a) Energy consumption in a small number of iterations. (b) Latency in a small number of iterations. (c) Task success rate in a small number of iterations.

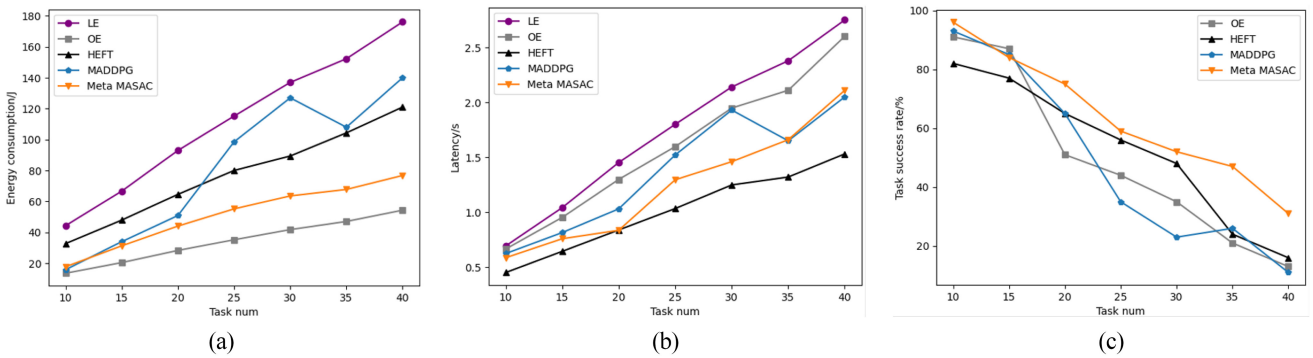



Fig. 8. Comparison of each strategy with respect to different metrics when the task number changes. (a) Energy consumption based on the variable task number. (b) Latency based on the variable task number. (c) Task success rate based on the variable task number.

consumption, latency, and task success rate. Its comprehensive performance is second only to the meta-MASAC-based strategy, which shows that the MADDPG algorithm can be applied to new DAGs with the same number of subtasks. However, the new task structure has an impact on its performance as well; hence, it has a mediocre performance in generalization. The meta-MASAC-based strategy has the best overall performance in all aspects and nearly converges at the beginning of the iteration, which indicates that the algorithm can be quickly adapted to the new task structure and has good performance in terms of convergence and generalization.

To compare the influence of the total subtasks' number on each algorithm strategy during the experiment, the range of this value was set to $[10, 40]$. Correspondingly, the step size is set to five and the arbitrary types of DAG tasks are generated for different task scales. As depicted in Fig. 8, with an increase in the total number of subtasks, the results of each strategy in terms of energy consumption and latency increase, resulting in a decrease in the task success rate. Among the compared methods, the LE-based strategy has the highest energy consumption and latency owing to its local execution only, which is limited by the performance and power of MDs. The OE-based strategy performs best in terms of energy consumption because it prioritizes the use of edge servers for task processing and allocates fewer tasks to MDs. However, as the load on the edge servers increases, the computational resources allocated to each subtask are correspondingly reduced. Thus, its latency gradually increases. The HEFT-based strategy prioritizes some tasks based on task completion time; hence, it performs the best in terms of latency. The MADDPG-based strategy cannot guarantee the optimal solution in all task types owing to the lack of generalization. Thus, it has average performance in terms of energy consumption, latency, and task success rate. The strategy based on the meta-MASAC algorithm has the best overall performance in all aspects and is highly adaptable to dynamic environments. Therefore, it can guarantee a better solution for different task sizes.

To further verify the effect of different environments on each strategy, the value range of the transmission rate was set to $[6, 10]$ with a step size of 0.5. As depicted in Fig. 9, with an increase in the transmission rate, the performance of all strategies in terms of energy consumption and latency is reduced; except for the LE strategy, which does not perform offloading execution at all. The corresponding task success rate is also improved. However, the MADDPG algorithm lacks the adaptability to different environments; hence, it obtains unstable results. The strategy based on the meta-MASAC algorithm can adapt well to the effects of different transmission rates, and its energy consumption and latency decrease to a certain extent with an increase in transmission rate. Its performance is the best among all strategies in terms of task success rate, which underlines the excellent performance of the algorithm and its ability to adapt to different environments.

Finally, this study examined the impact of this environmental factor on each algorithm by varying the computational performance of the edge servers. As presented in Fig. 10, if the CPU processing performance of the edge server increases, all aspects of the MADDPG algorithm change significantly.

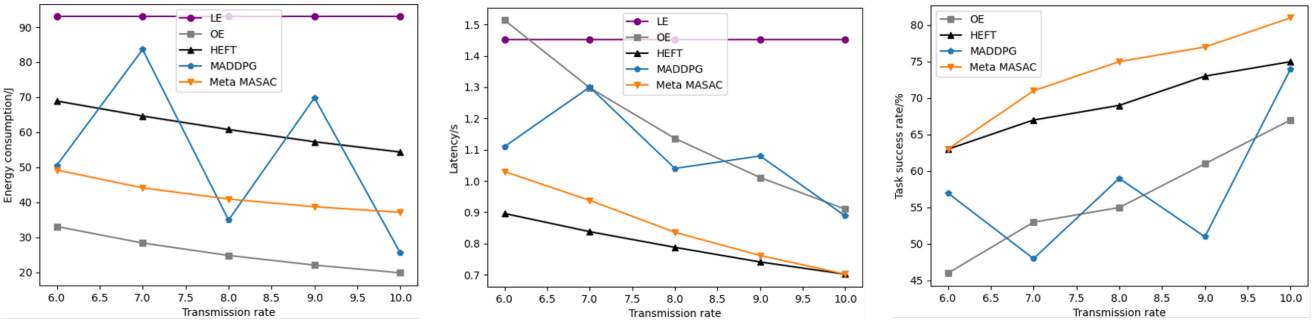



Fig. 9. Comparison of each strategy with respect to different metrics when the transmission rate changes. (a) Energy consumption based on the variable transmission rate. (b) Latency based on the variable transmission rate changed. (c) Task success rate based on the variable transmission rate.

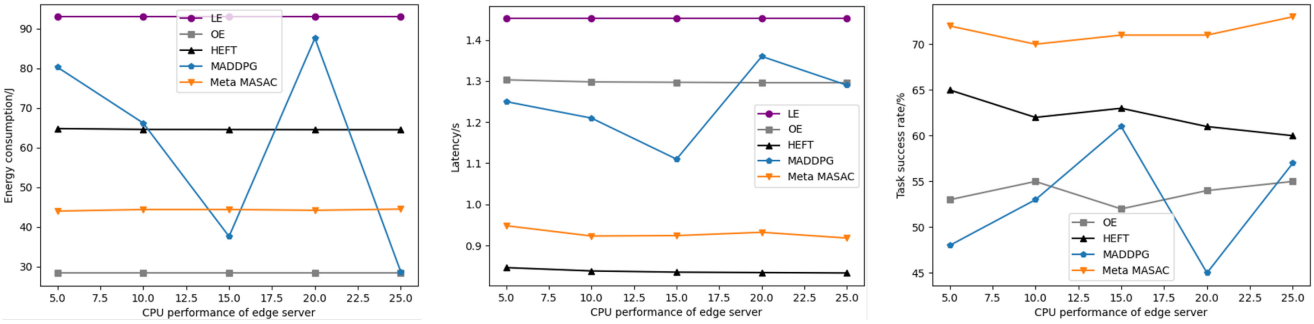



Fig. 10. Comparison of each strategy with respect to different metrics when the edge server performance changes. (a) Energy consumption based on the variable performance of the edge server. (b) Latency based on the variable performance of the edge server. (c) Task success rate based on the variable performance of the edge server.

The other algorithms remain unchanged, which indicates that the current performance of the edge server is sufficient to meet the processing requirements of the DAG tasks in the default environment. Therefore, we can conclude that there is little correlation between its offloading strategy and the edge server performance. Meanwhile, the performance of the MADDPG algorithm-based strategy is unstable in all aspects, which is mainly because of the lack of algorithm generalization. This results in large differences in the corresponding strategy when the environment changes, further reflecting the advantage of the meta-MASAC algorithm in adapting to dynamic environments.

## VII. Conclusion

A strategy based on multiagent MRL is proposed to address the problems of mobile applications with multiple DAG types and MDs with WPT services to ensure the quick generation of offloading strategies for different DAG task types. This study constructs a system architecture with task offloading and energy harvesting in a MEC environment, combines meta-learning with this architecture, and sets up a meta-learner and base learner. Then, it deploys them in the MD and the edge server, respectively, to fully leverage device characteristics and service requirements. In addition, this study splits the MRL process into four steps: 1) base learner training; 2) meta-information storage; 3) meta-learner training; and 4) meta-learner application, which are used to achieve fast learning through the collaborative operation between two learners. Subsequently, the MDP model based on a resource-limited MEC environment is constructed and combined with MRL algorithms to determine the WPT service duration and whether to offload the task. Finally, the experimental results indicate that the meta-MASAC-based strategy achieves the best overall performance with respect to all aspects, and can be quickly applied to various environments with good stability and generalization.

Although the meta-MASAC-based strategy has many benefits for the MEC model proposed in this study, several challenges require further exploration. In this study, we considered stable wireless channels, reliable MDs, and sufficient computational resources. Thus, the strategy does not break down when the number of users increases. However, some MDs may drop out owing to broken network connections or insufficient power. This situation can affect the training process of our strategy. In addition, the MEC environment comprises three parts: 1) cloud data centers; 2) edge servers; and 3) MDs. However, in this study, only the task offloading policy between the MD and the edge server is examined. Therefore, as a future research direction, we aim to further study the application of meta-algorithms in the cloud–edge–device architecture, edge-to-edge architecture, and device-to-device architecture.

## Acknowledgment

Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the above agencies.

## REFERENCES


[1] H. Ke, J. Wang, L. Deng, Y. Ge, and H. Wang, "Deep reinforcement learning-based adaptive computation offloading for MEC in heterogeneous vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7916–7929, Jul. 2020.

[2] B. Liu, X. Xu, L. Qi, Q. Ni, and W. Dou, "Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user MEC environment," *J. Syst. Archit.*, vol. 114, Mar. 2021, Art. no. 101970.

[3] J. Liang, K. Li, C. Liu, and K. Li, "Joint offloading and scheduling decisions for DAG applications in mobile edge computing," *Neurocomputing*, vol. 424, pp. 160–171, Feb. 2021.

[4] Z. Lv, L. Qiao, and A. K. Singh, "Advanced machine learning on cognitive computing for human behavior analysis," *IEEE Trans. Computat. Social Syst.*, vol. 8, no. 5, pp. 1194–1202, Oct. 2021.

[5] M. Monforte and F. Ficuciello, "A reinforcement learning method using multifunctional principal component analysis for human-like grasping," *IEEE Trans. Cogn. Devel. Syst.*, vol. 13, no. 1, pp. 132–140, Mar. 2021.

[6] Q. He, Z. Dong, F. Chen, S. Deng, W. Liang, and Y. Yang, "Pyramid: Enabling hierarchical neural networks with edge computing," in *Proc. ACM Web Conf.*, 2022, pp. 1860–1870.

[7] F. Luo, Q. Zhou, J. Fuentes, W. Ding, and C. Gu, "A soar-based space exploration algorithm for mobile robots," *Entropy*, vol. 24, p. 426, Mar. 2022.

[8] B. M. Smith, M. Thomasson, Y. C. Yang, C. Sibert, and A. Stocco, "When fear shrinks the brain: A computational model of the effects of posttraumatic stress on hippocampal volume," *Topics Cogn. Sci.*, vol. 13, no. 3, pp. 499–514, 2020.

[9] F. Jauro, H. Chiroma, A. Y. Gital, M. Almutairi, S. M. Abdulhamid, and J. H. Abawajy, "Deep learning architectures in emerging cloud computing architectures: Recent development, challenges and next research trend," *Appl. Soft Comput.*, vol. 96, Nov. 2020, Art. no. 106582.

[10] Y. Rizk, M. Awad, and E. W. Tunstel, "Decision making in multiagent systems: A survey," *IEEE Trans. Cogn. Devel. Syst.*, vol. 10, no. 3, pp. 514–529, Sep. 2018.

[11] D. Zhang et al., "Near-optimal and truthful online auction for computation offloading in green edge-computing systems," *IEEE Trans. Mobile Comput.*, vol. 19, no. 4, pp. 880–893, Apr. 2020.

[12] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, Aug. 2020.

[13] M. Li, T. Chen, J. Zeng, X. Zhou, K. Li, and H. Qi, "D2D-assisted computation offloading for mobile edge computing systems with energy harvesting," in *Proc. 20th Int. Conf. Parallel Distrib. Comput. Appl. Technol. (PDCAT)*, 2019, pp. 90–95.

[14] T. Zhang and W. Chen, "Computation offloading in heterogeneous mobile edge computing with energy harvesting," 2020, *arXiv:2004.08073*.

[15] M. Merluzzi, P. Di Lorenzo, and S. Barbarossa, "Latency-constrained dynamic computation offloading with energy harvesting IoT devices," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 750–755.

[16] C. Kai, H. Zhou, Y. Yi, and W. Huang, "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 2, pp. 624–634, Jun. 2021.

[17] J. Ren, G. Yu, Y. Cai, Y. He, and F. Qu, "Partial offloading for latency minimization in mobile-edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2017, pp. 1–6.

[18] W. Zhan, C. Luo, G. Min, C. Wang, Q. Zhu, and H. Duan, "Mobility-aware multi-user offloading optimization for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3341–3356, Mar. 2020.

[19] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.

[20] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Sep. 2021.

[21] C. Chen, Y. Zhang, Z. Wang, S. Wan, and Q. Pei, "Distributed computation offloading method based on deep reinforcement learning in ICV," *Appl. Soft Comput.*, vol. 103, May 2021, Art. no. 107108.

[22] H. Peng and X. Shen, "Multi-agent reinforcement learning based resource management in MEC-and UAV-assisted vehicular networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 131–141, Jan. 2021.

[23] S. Wen, Z. Wen, D. Zhang, H. Zhang, and T. Wang, "A multi-robot path-planning algorithm for autonomous navigation using meta-reinforcement learning based on transfer learning," *Appl. Soft Comput.*, vol. 110, Oct. 2021, Art. no. 107605.

[24] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5331–5340.

[25] D. Wang, B. Ding, and D. Feng, "Meta reinforcement learning with generative adversarial reward from expert knowledge," in *Proc. IEEE 3rd Int. Conf. Inf. Syst. Comput.-Aided Educ. (ICISCAE)*, 2020, pp. 1–7.

[26] J. Xu, L. Yao, L. Li, M. Ji, and G. Tang, "Argumentation based reinforcement learning for meta-knowledge extraction," *Inf. Sci.*, vol. 506, pp. 258–272, Jan. 2020.

[27] H. Jiang, D. Shi, C. Xue, Y. Wang, G. Wang, and Y. Zhang, "Multi-agent deep reinforcement learning with type-based hierarchical group communication," *Appl. Intell.*, vol. 51, pp. 5793–5808, Jan. 2021.

[28] 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, Jan. 2021.

[29] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[30] B. H. Abed-Alguni, D. J. Paul, S. K. Chalup, and F. A. Henskens, "A comparison study of cooperative Q-learning algorithms for independent learners," *Int. J. Artif. Intell*, vol. 14, no. 1, pp. 71–93, 2016.

[31] B. H. Abed-Alguni and M. A. Ottom, "Double delayed Q-learning," *Int. J. Artif. Intell.*, vol. 16, no. 2, pp. 41–59, 2018.

[32] Z. Yu, G. Xu, Y. Li, P. Liu, and L. Li, "Joint offloading and energy harvesting design in multiple time blocks for FDMA based wireless powered MEC," *Future Internet*, vol. 13, no. 3, p. 70, 2021.

[33] L. Wang, H. Shao, J. Li, X. Wen, and Z. Lu, "Optimal multi-user computation offloading strategy for wireless powered sensor networks," *IEEE Access*, vol. 8, pp. 35150–35160, 2020.

[34] Y. Hmimz, T. Chanyour, M. El Ghmary, and M. O. C. Malki, "Joint radio and local resources optimization for tasks offloading with priority in a mobile edge computing network," *Pervasive Mobile Comput.*, vol. 73, Jun. 2021, Art. no. 101368.

[35] S. Hu and Y. Xiao, "Design of cloud computing task offloading algorithm based on dynamic multi-objective evolution," *Future Gener. Comput. Syst.*, vol. 122, pp. 144–148, Sep. 2021.

[36] H. Lu, C. Gu, F. Luo, W. Ding, S. Zheng, and Y. Shen, "Optimization of task offloading strategy for mobile edge computing based on multi-agent deep reinforcement learning," *IEEE Access*, vol. 8, pp. 202573–202584, 2020.

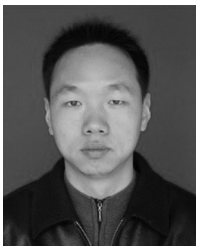
[37] D. Chen, Y.-C. Liu, B. Kim, J. Xie, C. S. Hong, and Z. Han, "Edge computing resources reservation in vehicular networks: A meta-learning approach," *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 5634–5646, May 2020.

[38] P. Lai et al., "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. Int. Conf. Serv. Orient. Comput.*, 2018, pp. 230–245.

[39] M. Merluzzi, P. Di Lorenzo, S. Barbarossa, and V. Frascolla, "Dynamic computation offloading in multi-access edge computing via ultra-reliable and low-latency communications," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 6, pp. 342–356, Mar. 2020, doi: 10.1109/TSIPN.2020.2981266.


**Weichao Ding** was born in 1989. He received the B.S. degree in computer science and technology from Northeast Forestry University, Harbin, China, in 2013. He is currently pursuing the M.S. and Ph.D. degrees in computer applications with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China.

His main research interests include cloud computing, cloud resource management and optimization, and big data applications.



**Fei Luo** was born in 1978. He received the B.S., M.S., and Ph.D. degrees in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1997, 2004, and 2008, respectively.

From 2008 to 2015, he was an Assistant Professor with the College of Information Science and Engineering, East China University of Science and Technology, Shanghai, China, where he has been an Associate Professor since 2015. He is the author of more than 30 papers and ten inventions. His research interests include distributed and cloud computing applications.

**Chunhua Gu** was born in 1970. He received the B.S., M.S., and Ph.D. degrees in computer science from East China University of Science and Technology, Shanghai, China, in 1992, 1998, and 2007, respectively.

He is a Professor and a Ph.D. Supervisor with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China. His main research interests include cloud computing and Internet of Things applications.

Dr. Gu is a Senior Member of the China Computer Federation.

**Zhiming Dai** was born in 1983. He received the master's degree in software engineering from Fudan University, Shanghai, China, in 2012. He is currently pursuing the Ph.D. degree with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai.

His main research interests include optimal scheduling of edge computing.

**Haifeng Lu** was born in 1993. He received the master's degree from the Computer Science Department, Donghua University, Shanghai, China, in 2017. He is currently pursuing the Ph.D. degree with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai.

His main research interests include edge computing and reinforcement learning applications.