Dynamic Offloading Based on Meta Deep Reinforcement Learning and Load Prediction in Smart Home Edge Computing

Mingchu Li(✉), Shuai Li, and Wanying Qi

School of Software Technology, Dalian University of Technology, Dalian 116620, China
mingchul@dlut.edu.cn

Abstract. In the edge computing enabled smart home scenario. Various smart home devices generate a large number of computing tasks, and users can offload these tasks to servers or perform them locally. Offloading to the server will result in lower delay, but it will also require paying the corresponding offloading cost. Therefore, users need to consider the low delay and additional costs caused by offloading. Different users have different trade-offs between latency and offload costs at different times. If the trade-off is set as a fixed hyperparameter, it will give users a poor experience. In the case of dynamic trade-offs, the model may have difficulty adapting to arrive at an optimal offloading decision. By jointly optimizing the task delay and offloading cost, We model it as a long-term cost minimization problem under dynamic trade-off (DT-LCMP). To solve the problem, we propose an offloading algorithm based on multi-agent meta deep reinforcement learning and load prediction (MAMRL-L). Combined with the idea of meta-learning, the DDQN method is used to train the network. By training the sampling data in different environments, the agent can adapt to the dynamic environment quickly. In order to improve the performance of the model, LSTNet is used to predict the load level of the next slot server in real time. The simulation results show that our algorithm has higher performance than the existing algorithms and benchmark algorithms.

Keywords: Edge computing · Task offloading · Meta deep reinforcement learning · Smart homes · Dynamic trade-off

1 Introduction

With the rapid development of the mobile Internet and the popularity of smart devices, the demand for large amounts of data generation and processing is increasing [1]. The traditional method of transmitting data to the cloud center for processing has faced some limitations and cannot fully meet people's increasing real-time and responsiveness requirements. Especially for application scenarios

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2024
Published by Springer Nature Switzerland AG 2024. All Rights Reserved
H. Gao et al. (Eds.): CollaborateCom 2023, LNICST 561, pp. 421–439, 2024.
https://doi.org/10.1007/978-3-031-54521-4_23

such as smart homes, high latency, strong dependencies, and privacy and security issues have become major challenges, and traditional cloud computing has been unable to solve these problems [2]. Edge computing has become a new paradigm to solve these problems. Unlike cloud computing, edge computing is closer to users and data sources, and can provide users with nearby services [3], process user data promptly, reduce costs, and reduce bandwidth pressure.

In smart homes, it is an inevitable trend to combine infrastructure with artificial intelligence (AI) [4]. The combination of the two will produce a variety of smart devices, such as smart shoe cabinets [5]. These smart devices involve AI functions in six key areas, namely activity perception, data processing, speech recognition, image cognition, decision making, and future prediction [6]. For example, in terms of image recognition, monitoring, and analyzing human activities and body characteristics to achieve facial, emotional, biological, and other recognition, and also to understand different scenarios. These tasks are usually latency-sensitive and computation-intensive, and require certain computing resources for processing. Long-term heavy computation can reduce the service life of Internet of Things (IoT) devices [7]. Offloading these tasks to edge servers is a more appropriate solution. This approach can effectively reduce not only the execution delay of tasks but also the maintenance cost of IoT devices. But it will also be accompanied by an additional increase in bandwidth delay [8] and a certain offloading cost [9].

This paper focuses on the joint optimization problem between delay and offloading cost when offloading tasks in edge computing-enabled smart home scenarios. Consider the user's dynamic trade-off between delay and offloading cost, as a dynamic parameter that can be changed by the user's input, rather than a fixed hyperparameter of the model. In reality, different users have varying measurement standards for delay and offloading costs at different times. For instance, freelancers at home might prioritize low latency, as they require efficient real-time task processing. For office workers outside, the delay requirements at home may not be so high. They just rest at home and do not need to perform frequent smart tasks. For the same user, when they are at home, the delay requirement may be higher, because they want to respond to smart home devices in a timely manner. Conversely, when they are outside, the latency requirements may be relatively lower since specific intelligent tasks continue even in their absence, such as smart security and energy management.

In this work, we focus on the joint optimization problem between latency and offload cost under user dynamic trade-offs. We reformulate it as a Markov decision process (MDP) and propose an offloading algorithm based on multi-agent meta deep reinforcement learning. Multi-agent meta deep reinforcement learning is an extended multi-agent deep reinforcement learning method whose goal is to allow agents to learn from multiple tasks and form a global policy for adapting quickly to new tasks. By combining meta-learning, the agent can learn the optimal strategy under different trade-off criteria, so that the agent can make decisions more flexibly and efficiently without retraining [14]. In this paper, a model-free meta-learning algorithm [15] is used to train the model, so that it has stronger adaptability and generalization ability in the face of different tasks and

environments, and achieves the effect of rapid convergence. Ability to quickly make adaptive decisions based on the current task situation and user trade-offs. Such an approach can not only improve the efficiency and performance of agents but also bring better user experience to smart home systems (SHS). The algorithm in this paper runs on multiple edge servers and multiple SHSs. When each SHS can observe server information but cannot observe the other's information, it trains a deep reinforcement learning agent to offload tasks. The main contributions of this paper are as follows:

1) We formulate the long-term cost minimization problem under dynamic trade-offs. This problem considers the joint optimization between the delay and offloading cost of task offloading and proposes a dynamic weighted sum to minimize the delay and offloading cost.
2) We propose the offloading algorithm MAMRL-L. The algorithm is based on multi-agent deep reinforcement learning, and uses DDQN to train agents, so that each agent can perform efficient task offloading when only server information can be observed, but other agents cannot be observed. At the same time, combined with the meta-learning method, it is proposed to use probability sampling to train the meta-policy to improve the adaptability of the agent in the dynamic environment. To improve the performance of the algorithm, we combine LSTNet [39] to predict the future server information to optimize the offloading decision.
3) Through simulation experiments, we compare the DMRO and the benchmark algorithm. The results show that our proposed algorithm achieves efficient offloading of computing tasks, significantly reduces the long-term cost of users, and can adapt quickly to the new environment.

The other parts of this article are arranged as follows. Section 2 discusses the related work, and Sect. 3 introduces the system model and problem description. In Sect. 4, the offloading algorithm based on multi-agent deep reinforcement learning and LSTNet is introduced in detail, and the optimal offloading scheme of computing tasks is obtained by this algorithm. In Sect. 5, the performance of the proposed algorithm is evaluated based on simulation experiments, and other algorithms are compared. Section 6 summarizes the work of this paper and looks forward to future research.

2 Related Works

In the current research, a wealth of research results have emerged for the problem of task offloading in the mobile edge computing (MEC) environment. From different perspectives, these studies are dedicated to solving multiple challenges in MEC systems.

2.1 User Experience

In terms of guaranteeing user experience (QoE). Based on traditional algorithms, Jiang et al. [16] proposed an online joint offloading and resource allocation method under long-term MEC energy constraints to solve the problems

of increased energy consumption and poor resource allocation caused by task offloading. At the same time, Lyapunov is used to pursue the optimization of long-term user experience (QoE). Luo et al. [17] solved offloading decisions by allocating communication and computing resources jointly. The offloading problem and deep reinforcement learning (DRL) have become research hotspots in recent years. Zhou et al. [19] redefined the QoE metric and used the DRL algorithm to make optimal offloading decisions. Considering that in the limited experience in the distributed environment, the time state of the fully connected layer is not considered by the existing DRL method, Park et al. [18] proposed a calculation offloading scheme based on distributed DRL.

2.2 Delay and Energy Consumption

Delay and energy consumption constraints are the focus of many researchers. Reference [10–12] focused on minimizing system delay and energy consumption in task offloading. Huang et al. [13] studied joint task offloading decision and bandwidth allocation optimization to minimize the overall offloading cost in terms of energy cost, computing cost, and delay cost, and proposed an MEC task offloading and resource allocation algorithm based on Double Deep-Q network (DDQN). To minimize the task computing delay of single-user multi-edge server system and balance the workload between edge servers, the optimization algorithm based on downlink NOMA is studied in [20]. To solve the problem of limited battery capacity and low computing power of IoT nodes, Seo et al. [21] used a deep neural network (DNN) and corresponding exploration and training strategies to learn near-optimal wireless power transmission (WPT) duration. In the task offloading scheduling in dynamic MEC systems, Wang et al. [22] combined energy harvesting technology with IoT devices to provide a hybrid energy supply model, which minimizes the cost of the system.

2.3 Pricing Strategy

Some papers focus on solving the problem of MEC server pricing in edge environments. Considering the imbalance of information availability, Chen et al. [23] introduced the Stackelberg game to describe the relationship between the MEC server and the end users (EUs) to obtain a reasonable pricing scheme. In terms of differential pricing, Chen et al. [24] determined the unit price according to the actual usage of computing resources. According to the obtained unit price and execution delay, the Stackelberg game method is developed to obtain the user's optimal offloading strategy and the server's equilibrium pricing strategy. In addition to static pricing, dynamic pricing is also a problem worth studying. Xu et al. [25] studied the dynamic pricing problem that meets the computing needs of users that change over time, to minimize the scheduling cost of servers and maximize the service pricing utility of service providers. Developed a decentralized online optimization framework. Considering the task offloading needs of resource-poor users, Qu et al. [26] proposed a DNN combined with a baseline neural network BNN as a policy network to design price policies.

2.4 Dynamic Offloading

To bring higher performance and lower latency in various scenarios. In [27], a dynamic task offloading (DDTO) algorithm based on deep reinforcement learning (DRL) was proposed considering the rapidly increasing problem size and solution space size under changing channel conditions and task generation. In order to incorporate renewable resources into mobile edge computing while efficiently managing resources, Yan et al. [28] proposed a reinforcement learning-based algorithm that can dynamically learn offloading schemes and optimal provisioning strategies for edge servers. In order to quickly adapt to a dynamic environment, Dabin et al. [29] proposed a meta deep reinforcement learning-based offloading (DMRO) algorithm. The algorithm aggregates the perception ability of deep learning, the decision-making ability of reinforcement learning, and the rapid environment learning ability of meta-learning. Cai et al. [30] proposed a training method for meta-reinforcement learning, including local training based on a specific task policy and meta-policy-based training on MEC.

These studies consider the problems existing in edge computing systems from different perspectives. However, in the process of task offloading, the dynamic trade-off between delay and offloading cost has not been considered. This question mainly studies and solves this problem.

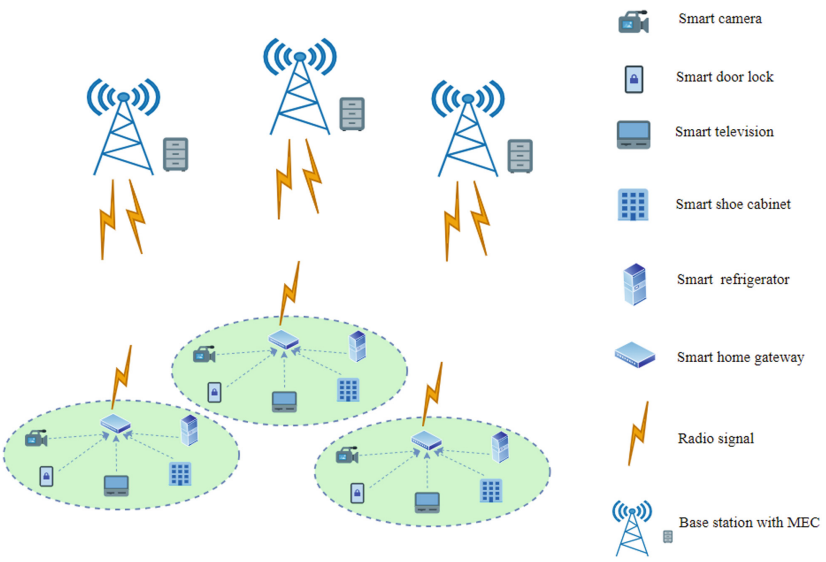


Fig. 1. Model diagram of smart home combined with edge computing.

3 System Model and Problem Formulation

We consider that in a multi-smart home system with multiple edge servers, $\mathcal{N}$ = { 1,2,..., N } is used to represent the set of N smart home systems (SHS), and $\mathcal{M}$ = { 1,2,..., M } is used to represent the set of M base stations (BS) equipped with edge servers. The BS, edge server, and server mentioned later in this article have the same meaning. As shown in Fig. 1, each SHS has multiple smart devices (SDs) and a smart gateway (SG). SG can bridge the internal and external networks of the home, transform different communication methods, and collaboratively manage various SDs equipped in the smart home [31]. In this paper, SG is responsible for processing data generated by various SDs and decides to compute locally or offload to BS for processing. We assume that the rate of receiving a task in the SG is R, and the length of the time slot τ is defined as 1/R, that is, each time slot generates a task. Then $\mathcal{T}$ = { 1,... T } is used to represent the set of T time slots. Task V is represented by a tuple { d, c }, where d represents the data size of the task (in bits as a unit), and c represents the number of CPU cycles required to process per-bit data. p represents the price to be paid for each time slot occupied by the server. In addition, we define $n \in \mathcal{N}$ to represent the n^{th} SG, $m \in \mathcal{M}$ to denote the m^{th} BS, and $t \in \mathcal{T}$ to denote the t^{th} slot. The t time slot and the t^{th} time slot have the same meaning. Table 1 shows the main notations.

When transmitting over a wireless channel, the signal will experience effects such as signal attenuation and channel gain during transmission [32]. Signal attenuation means that the signal strength decreases with the increase of transmission distance, while channel gain means that the signal is affected by the channel during transmission, and the signal may be strengthened or weakened. In order to better describe the transmission rate between devices, we express the transmission rate of SG as:

$$r_m = B \log_2 \left(1 + \frac{P h_m}{\sigma^2}\right) \tag{1}$$

Among them, r_m represents the transmission rate between the SG and the BS m. B and P represent the bandwidth and transmission power of the SG, respectively, and h_m represents the channel gain between the SG and the BS m. σ^2 represents the noise power. If the task is executed locally, c_t and d_t represent the number of CPU instructions per bit and the data size of the task in the t^{th} time slot, respectively, the local execution delay of the task processed by the SG in the t^{th} time slot is as follows:

$$D_t^{ex} = \frac{c_t \cdot d_t}{f_{local}} + D_{local,t}^{wait\_ex} \tag{2}$$

Here, $D_{local,t}^{wait\_ex} = \sum_{i=1}^{u'} \frac{c_i \cdot d_i}{f_{local}}$ represents the time that the task at time t is waiting for execution locally, and u′ represents the number of tasks to be executed in the current local execution queue. If the task is offloaded to the BS for execution, the task data needs to be uploaded to the BS. The transmission delay of the task in the t slot is as follows:

Table 1. Notations.

Notations	Definition
$\mathcal{M}$={1,2,...M}	Set of the MEC
$\mathcal{N}$={1,2,...N}	Set of the SHS
R	The generation rate of task
$\mathcal{T}$={1,2,...T}	Set of the time slot
τ	Length of time slot
λ_t	The weight of the t^{th} time slot
$\Lambda = \{\lambda_t, t \in \mathcal{T}\}$	Set of the weight of each time slot
V(d,c)	The data size of the task and the number of CPU cycles required per-bit of data
r_m	The transmission rate between the SG and the m^{th} base station
B	The bandwidth of SG
P	The transmission power of SG
σ^2	The noise power
$D_{ex}^{local,t}$	The local execution delay of the task in t^{th} time slot
$D_{local,t}^{wait\_ex}$	The local execution wait delay of the task in t^{th} time slot
D_t^{tr}	The transmission delay of the task in t^{th} time slot
$D_t^{wait_{tr}}$	The local transmission wait delay of the task in t^{th} time slot
$D_{m,t}^{wait}$	The server wait delay of the task in t^{th} time slot
$D_{m,t}^{ex}$	The server execution delay of the task in t^{th} time slot
f_{local}	CPU frequency of local device
f_m	CPU frequency of the m^{th} server
x_t	Indicates whether the task is offloaded. 0 or 1
$y_t \in \{1, 2, ..., M\}$	Offload to which server to execute
D_t^{total}	The total delay of the task in t^{th} time slot
p	The price to be paid to occupy a server time slot
$cost_t$	The cost of the task in t^{th} time slot
s	The environmental state observed by the agent
a	The action given by the agent
r	The reward for the action given by the agent is by state s

$$D_t^{tr} = \frac{d_t}{r_m} + D_t^{wait\_tr} \tag{3}$$

where, $D_t^{wait\_tr} = \sum_{i=1}^{q'} \frac{d_i}{r_m}$ represents the waiting delay in the transmission queue, and q′ represents the number of tasks to be transmitted in the current transmission queue. Since the calculation result of the task is usually much smaller than the task data size, the downlink delay of the calculation result is ignored [33]. The main frequency of the server is generally much higher than that

of the local device, and it is multi-core and multi-threaded. Under the default CPU trade-off [34] tuning scheme, the task can be run on multiple cores. Using f_m to represent the CPU frequency of server m, the execution delay of the task on server m is as follows:

$$D_{m,t}^{ex} = \frac{c_t \cdot d_t}{f_m} \tag{4}$$

Without considering the task priority, the waiting delay of the task on the server m is equal to the sum of the execution delay of the tasks to be executed in the server m, and n′ is used to represent the number of tasks currently to be executed in the server, then the waiting time of the task Extension is expressed as:

$$D_{m,t}^{wait} = \sum_{i=1}^{n'} \frac{c_i \cdot d_i}{f_m} \tag{5}$$

Then the total delay of the task on the server is: $D_{m,t}^{total} = D_{m,t}^{ex} + D_{m,t}^{wait}$. Define x_t as an indicator, $x_t \in \{0, 1\}$. 0 represents local execution, and 1 represents offloading to BS. The total delay of the task as:

$$D_t^{total} = (1 - x_t) \cdot D_t^{ex} + x_t \cdot (D_t^{tr} + D_{m,t}^{total}) \tag{6}$$

Under the traditional billing method, there will be problems such as waste of resources, lack of flexibility, and difficulty in estimating costs. Pay-as-you-go has gradually become a more reasonable payment model [35], which has the advantages of flexibility, cost reduction, and changing business needs [36]. We use p to denote the price to be paid per slot of server, then the offloading cost of the task generated by t^{th} time slot can be expressed as:

$$cost_t = D_{m,t}^{ex} \cdot p \cdot x_t \tag{7}$$

If the task is executed locally at $x_t = 0$, the $cost_t$ of offloading is 0. We consider the long-term cost minimization problem under dynamic trade-offs, that is, the dynamic weighted sum problem of joint optimization of delay and offloading cost. Then the optimization objective is expressed as:

$$DT - LCMP : min \sum_{t=1}^{T} ((1 - \lambda_t) \cdot cost_t + \lambda_t \cdot D_t^{total}) \tag{8}$$

This is an NP-hard problem. We use $\Lambda = \{\lambda_t, t \in \Im\}$ to represent the user's weight set, where $\lambda_t \in [0, 1]$ and represents the weight of t^{th} time slot. Without loss of generality, we take the step size of λ to be 0.1.

4 Algorithm Based on Meta Deep Reinforcement Learning and Long-Term and Short-Term Time Series Networks

In this section, we describe the establishment of MDP in detail and propose a task offloading algorithm based on multi-agent meta deep reinforcement learning and a load forecasting algorithm based on LSTNet, Combining MLP with LSTNet improves the performance and convergence speed of model training.

4.1 MDP Description

For the long-term cost minimization problem under dynamic trade-offs proposed in this paper, we transform it into a partially observable Markov decision process. Multiple smart home systems (SHS) with multiple base stations (BS) equipped with edge servers, each SHS agent observes the state of the local device and the state of the edge servers, and takes action to enter the next state, which is expressed as a multi-agent Markov decision process (MAMDP) [37], using tuples ($\mathcal{U}$, $\mathcal{S}$, $\{A_n\}_{n\in\mathcal{U}}$, $\mathcal{P}$, $\{R_n\}_{n\in\mathcal{U}}$, γ). Where $\mathcal{U}$ represents the set of agents, $\mathcal{S}$ represents the set of state of all agents, A_n represents the set of actions of agents, $\mathcal{P}$ represents the probability of transferring to the next state, R_n represents the set of rewards after the agent takes action, and γ represents the discount factor. Each SHS trains an agent in the SG, and the agent interacts with the environment to continuously learn and optimize offloading decisions. The agent state, action, and reward at time slot t are defined as follows:

- State: s_t is defined as $\{d_t, c_t, SG_t, \{r_m, I_t\}_{m\in\mathcal{M},t\in\mathcal{T}}\}$. Among them, d_t and c_t represent task information, and SG_t represents local device information, including transmission queue information, load information (the current time slot being processed number of tasks), and CPU frequency. r_m indicates the transmission speed with the edge server, and m_t indicates the CPU frequency and load information of the edge server. s_{t+1} represents the state at the next moment.
- Action: a_t represents the action chosen for time t. The action space $a_t \in \{0, 1, ...,\mathrm{M}\}$. Among them, 0 means to offload to the local, $\{1, 2, ...,\mathrm{M}\}$ means to offload to which server.
- Reward: r_t represents the reward for the agent to take actions in state s_t. For the convenience of exposition, we transform DT-LCMP into the reward maximization problem R-DT-LCMP, where R-DT-LCMP is expressed as

$$R - DT - LCMP : max \sum_{t=1}^{T} -((1-\lambda_t)\cdot cost_t + \lambda_t \cdot D_t^{total}) \tag{9}$$

4.2 Task Offloading Based on Multi-agent Meta Deep Reinforcement Learning

Multi-agent meta deep reinforcement learning is a method that combines meta-learning and multi-agent deep reinforcement learning to improve the agent's ability to use in dynamic environments. The meta-strategy improves the rapid adaptability in dynamic environments by collecting samples from each environment to train the model. In this paper, we use meta-policy gradients to implement meta deep reinforcement learning and use the DDQN method for training. Meta deep reinforcement learning includes inner and outer loops:

- Internal loop: We use DDQN [38] as the training method of the internal loop. DDQN is a value function optimization method, which optimizes the strategy by learning an action-value function (Q function). The update of Q function

is expressed as $Q'(s, a_t) = Q(s, a_t) + \gamma \max_{a_t} \hat{Q}(s_{t+1}, a_t) \cdot done_t$, where Q' represents the updated Q function, $\hat{Q}$ represents the target Q function, γ represents the discount factor, $done_t$ represents whether the last time slot is reached, and the arrival $done_t$ is equal to 0, otherwise, it is 1. For each agent, we use MLP to build a neural network model, use the DDQN algorithm to update the parameters of the MLP network and select actions based on the ϵ-greedy strategy, as shown in formula 10. In order to improve the sample efficiency, the experience playback mechanism is used to reuse the previous data for sampling training. The structure and training process of the DDQN model are shown in Fig. 2

$$a_t = \begin{cases} \text{argmax}_{a_t} Q(s_t, a_t), & \text{random number} > \varepsilon \\ \text{random action } a_t, & \text{random number} \leq \varepsilon \end{cases} \tag{10}$$

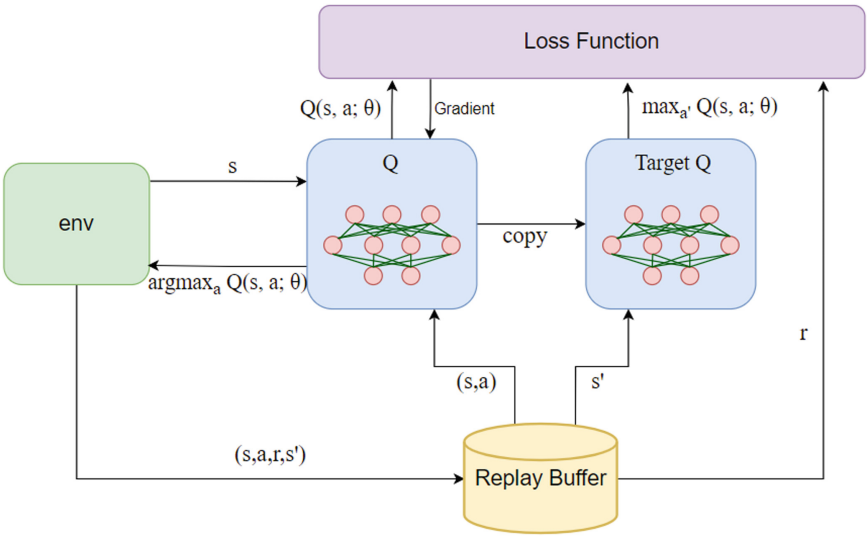


Fig. 2. DDQN model structure and training process.

– Outer loop: Consider establishing the task offloading process by each weight as an MDP, and the meta-policy treats each MDP as a training task. Assuming that all MDP tasks obey the distribution L, the meta-policy designed in this paper samples MDP according to the probability $\mathcal{P}$(i) of the distribution L, and performs training to obtain the loss. $\mathcal{P}$(i) represents the MDP sampling frequency corresponding to the i^{th} weight. This allows the model to learn more about the MDP with higher probability. Specifically, we train the i^{th} MDP according to the probability $\mathcal{P}$(i) and calculate the loss, such as the formula 11.

$$loss_i' = \begin{cases} 0, & \text{random number} > \mathcal{P}(i) \\ loss_i, & \text{random number} \leq \mathcal{P}(i) \end{cases} \tag{11}$$

If the random number is less than the frequency, the sampling is successful, and the loss of the i^{th} MDP is obtained. Otherwise, the sampling fails and the loss is 0. On this basis, the loss of the outer loop is calculated using the average error and Loss is expressed as formula 12.

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N} loss_i' \tag{12}$$

Algorithm 1. Multi-agent Deep Reinforcement Learning Algorithm

Initialize the number of agents, M; environment, env; model parameters; time steps, T; the number of episode, E; replay buffer rb with size M; Sampling probability, P.
while e from 1 to E **do**
    Reset env
    Observe the state of the environment, get the state
    Initialize $loss_{set}$
    while t from 1 to T **do**
        while m from 1 to M **do**
            According the formula 10, choose the action at
            Execute at, observe $next_{state}$ st, reward rt and $done_t$.
            store D$(s_t, a_t, r_t, s_{t+1}, done_t)$ in rb_m
        end while
        while m from 1 to M **do**
            sample batches of tuples uniformly from rb_m, called Db
            According to the formula 11, using Db to calculate, get the loss
            if random number $< p_m$ **then**
                push loss to $loss_{set}$
            else
                push 0 to $loss_{set}$
            end if
        end while
        Calculate the average error of the $loss_{set}$, called Loss
        Update model with Loss
    end while
end while

In the training phase, the inner loop is trained on the SG, and by collecting the samples of the inner loop training process, the model on the BS performs probability sampling on each SG sample to perform the outer loop training. In the test phase, the meta-policy parameters trained on the BS are updated to each SG, and each SG generates a weight set that obeys the beta distribution by setting different beta distribution parameters. In a dynamic environment where weights change over time, evaluate the model's performance and convergence rate by observing how the total reward changes. According to the description of the inner loop and the outer loop, we express the training process of the multi-agent meta deep reinforcement learning model as Algorithm 1.

4.3 Load Forecasting Based on Long-Term and Short-Term Time Series Networks

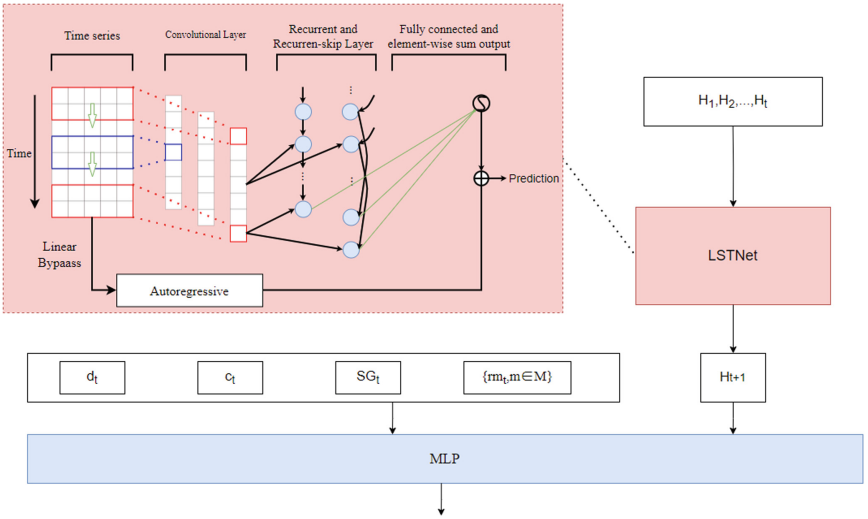


Fig. 3. The Network Model of MLP combined with LSTNet.

When the agent makes an unloading decision, predicting the future load situation through the historical information of the edge server can effectively improve the performance and convergence speed of the model [11]. Considering the mutual influence between servers and the long-term dependence of individual server information, the LSTNet [39] model based on a spatiotemporal attention mechanism is a good choice. The self-attention mechanism adaptively weights different positions in the input sequence, so as to better handle long-term dependencies and avoid information loss in the process of information transmission. The LSTNet model includes a Convolutional Layer, Recurrent and Recurrent-skip Layer, and fully connected and element-wise sum output. The Convolutional Layer extracts information in the time dimension and local dependencies between variables. Recurrent and Recurrent-skip Layer can effectively process time series data, capture long-term dependencies, and improve the training efficiency and generalization ability of the model. The final prediction result is obtained by element-wise summation through fully connected and element-wise sum output, and the autoregressive model is used as a linear component to solve the problem that the output scale of convolution and loop components is not sensitive to different input scales.

The combined model of MLP and LSTNet is shown in Fig. 3. We use the server's historical load information $\mathrm{H} = \{H_1, H_2, ...H_t\}$ to predict the load information H_{t+1} at the next moment. Replace the server information in the input

state s_t of DDQN with the prediction information $\{H_{m,t}\}, m \in \mathcal{M}, t \in \mathcal{T}$ of multiple servers. Therefore, LSTNet takes the input H and gets the output H_{t+1}. Since LSTNet must have certain historical data to start prediction, we define that when the amount of data reaches the input width of LSTNet, start training and forecasting.

5 Simulation Results

In this section, we first introduce the setting of experimental parameters. The performance of the proposed algorithm is evaluated by simulation experiments. And compared with other algorithms.

5.1 Parameters Setting

We consider a simulation experiment in a system consisting of multiple smart home systems (SHS) and multiple edge servers, each SHS contains an SG and other multiple smart devices. The tasks of all smart devices are uniformly processed at the SG. Considering that different smart devices generate different tasks at the same time, we set R to represent the generation rate of tasks, and the length of the time slot τ is set to $1/R$. The parameter settings of this experiment are shown in Table 2 with reference to the parameter settings of [40–42].

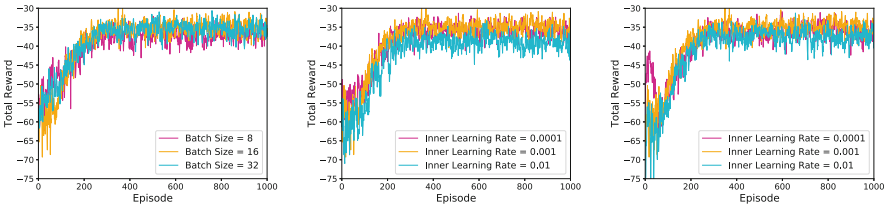
5.2 Performance Evaluation

We study the performance and convergence speed of the proposed MMRL-L algorithm under different parameters. Considering 1000 episodes, each episode contains 100 slots, our evaluation index is to evaluate the long-term rewards for each 100 slots.

In Fig. 4, we show the convergence speed of the model by different batch sizes and learning rates. In Fig. 4(a), with the increase of Episode, the model by the three parameters begins to converge at 300 episodes. When the Batch Size is 16, the final effect of the model is slightly higher than in the other two cases. In Fig. 4(b), we show the convergence of the model by different inner loop learning rates. It is not difficult to find that when the Inner Learning Rate is 0.01, due to the high learning rate, the model oscillates at the local optimal point, and the effect is lower than the other two cases. Finally, the inner Learning Rate is 0.001. Figure 4(c) shows the different performance and convergence speeds of the model by different outer loop learning rates. Among them, the total reward of Outer Learning Rate of 0.001 is the largest, and it reaches near -35 when it converges.

Table 2. Algorithm Parameters.

Parameters	Values
M	4
N	50
T	100
R	Generate 5 tasks per s
τ	1/R
λ	0–1 Bera distribution,step is 0.1
d	300–500 KB uniformly
c	100–300 cycle/Bit uniformly
f_{SG}	1.0–1.5 GHz uniformly
f_{server}	4.0–6.0 GHz uniformly
the number of SG′CPU	4
the number of BS′CPU	8
B	20 MHz
p	0.2
γ	0.9
Batch size of Double Dqn	32
Learning rate for inner loop	0.001
Learning rate of outer loop	0.001



(a) Batch Size (b) Inner Learning Rate (c) Outer Learning Rate

Fig. 4. Model diagram of smart home combined with edge computing.

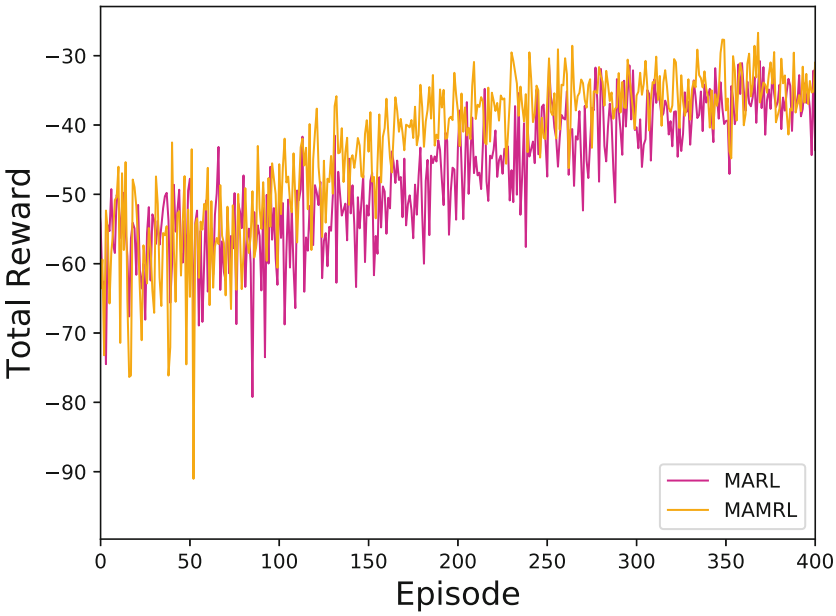


Fig. 5. Convergence of MARL and MAMRL.

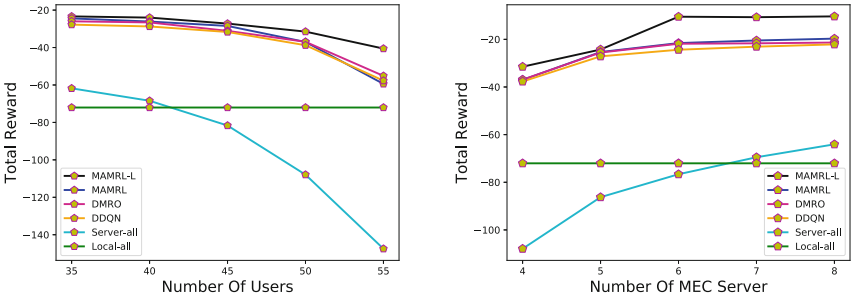
In Fig. 5, we observe the convergence of the multi-agent deep reinforcement learning model (MARL) and the multi-agent meta deep reinforcement learning model (MAMRL) in the dynamic environment. It is not difficult to find that the model converges faster after combining meta-learning.

5.3 Method Comparison

To verify the performance of the MAMRL-L algorithm, we compare it with the DMRO and four benchmark algorithms in a dynamic trade-off environment. All algorithms are as follows:

- Local-all: All tasks are executed locally without uninstalling
- Greedy-all: All tasks are offloaded to the server for execution. According to the greedy strategy, the server with the lowest load is selected according to the current observation state of the server by the local device.
- DDQN: DDQN(Double DQN) algorithm improves DQN, reduces overestimation problems, enhance performance and convergence speed, and more accurately evaluates action value.
- DMRO: An offloading algorithm based on meta deep reinforcement learning. When the proposed algorithm trains the meta-policy, it randomly selects a sample of one environment from multiple environments for learning at a time. After the learning is completed, the sample of another environment is selected for learning. The initial parameters of the model are obtained.

– MAMRL: The multi-agent deep reinforcement learning algorithm proposed in this paper. Compared with DMRO, we use probabilistic sampling for various MDP tasks, use the average error of sampling data for calculation, and use DDQN method for model training.
– MAMRL-L: An offloading algorithm is proposed based on multi-agent deep reinforcement learning and LSTNet in this paper.



(a) Number of Users (b) Number of MEC Servers

Fig. 6. The impact of the number of users or the number of MEC Servers.

We show the impact of the number of users or servers, as shown in Fig. 6. In Fig. 6(a), We show the total rewards for different numbers of users. As the number of users increases, the pressure on the server increases, resulting in a backlog of tasks and an increase in the execution delay of subsequent offloading tasks. The Local-all algorithm performs all tasks locally, so when the server increases, there will be no change. Other algorithms reduce the final total reward due to the increase in delay, especially Server-all. This is because only offloading tasks to the server will lead to an increase in the waiting delay of tasks, and will affect the waiting delay of all tasks that are later offloaded. It can be seen that the performance of DDQN, DMRO, and MMRL algorithms is similar. Citation meta-learning does not directly improve the performance of the model but improves its environmental adaptability. Because MAMRL-L combines the LSTNet algorithm, has the load information of the future server and optimizes the offloading decision, and the final effect is the best.

Figure 6(b) shows that as the number of edge servers increases, the total reward gradually increases, because the increased servers share the pressure of other servers and reduce the waiting delay of tasks. When the number of servers increases from 7 to 8, the waiting delay for tasks is very small, and adding more servers can not significantly reduce the waiting delay. It can be found that MAMRL-L has better results than other algorithms.

6 Conclusion

In this paper, we study the scenario of edge computing-enabled smart homes, considering the dynamic weighted sum of task execution delay and offloading cost by dynamic trade-offs. Based on this, the long-term cost minimization problem by dynamic trade-off is formulated. To solve this problem, an offloading algorithm based on multi-agent deep reinforcement learning and long-term and short-term time series network is proposed, which can quickly adapt to different environments and obtain the optimal task offloading strategy. Compared with other algorithms, our algorithm effectively reduces the long-term cost of task offloading in a dynamic trade-off environment.

In future work, we will consider more factors that have an impact on the offloading algorithm, including energy consumption, pricing mechanisms of different service providers, and bandwidth allocation issues. Furthermore, we will focus on the optimization problem of combining task offloading and task caching.

Acknowledgments. This paper is supported by the National Nature Science Foundation of China under grant number: T2350710232.

References

1. Naeem, M., et al.: Trends and future perspective challenges in big data. Smart Innov., Syst. Technol. **253**, 309–325 (2022)
2. Durao, F., et al.: A systematic review on cloud computing. J. Supercomput. **68**(3), 1321–1346 (2014)
3. Cao, K., et al.: An overview on edge computing research. IEEE Access **8**, 85714–85728 (2020)
4. Kopytko, V., et al.: Smart home and artificial intelligence as environment for the implementation of new technologies. Path Sci. **4**(9), 2007–2012 (2018)
5. Huh, J.H., Seo, K.: Artificial intelligence shoe cabinet using deep learning for smart home. In: Park, J., Loia, V., Choo, K.K., Yi, G. (eds.) Advanced Multimedia and Ubiquitous Engineering. Lecture Notes in Electrical Engineering, vol. 518, pp. 825–834. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-1328-8_108
6. Guo, X., et al.: Review on the application of artificial intelligence in smart homes. Smart Cities **2**(3), 402–420 (2019)
7. Jeon, Y., et al.: Mobility-aware optimal task offloading in distributed edge computing. In: International Conference on Information Networking, 2021-January, pp. 65–68 (2021)
8. Wu, Y., et al.: Noma-assisted multi-access mobile edge computing: a joint optimization of computation offloading and time allocation. IEEE Trans. Veh. Technol. **67**(12), 12244–12258 (2018)
9. Zhang, T.: Data offloading in mobile edge computing: a coalition and pricing based approach. IEEE Access **6**, 2760–2767 (2017)
10. Huang, X., et al.: Vehicle speed aware computing task offloading and resource allocation based on multi-agent reinforcement learning in a vehicular edge computing network. In: Proceedings - 2020 IEEE 13th International Conference on Edge Computing, EDGE 2020, pp. 1–8 (2020)

11. Tu, Y., et al.: Task offloading based on LSTM prediction and deep reinforcement learning for efficient edge computing in IoT. Future Internet **14**(2), 30 (2022)
12. Zhou, H., et al.: Energy efficient joint computation offloading and service caching for mobile edge computing: a deep reinforcement learning approach. IEEE Trans. Green Commun. Netw. **7**(2), 950–961 (2023)
13. Huang, L., et al.: Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. Digit. Commun. Netw. **5**(1), 10–17 (2019)
14. Beck, J., et al.: A survey of meta-reinforcement learning (2023)
15. Finn, C., et al.: Model-agnostic meta-learning for fast adaptation of deep networks (2017). https://proceedings.mlr.press/v70/finn17a.html
16. Jiang, H., et al.: Joint task offloading and resource allocation for energy-constrained mobile edge computing. IEEE Trans. Mob. Comput. (2022)
17. Luo, J., et al.: QoE-driven computation offloading for edge computing. J. Syst. Architect. **97**, 34–39 (2019)
18. Park, J., Chung, K.: Distributed DRL-based computation offloading scheme for improving QoE in edge computing environments. Sensors **23**(8), 4166 (2023)
19. Zhou, Z., et al.: QoE-guaranteed heterogeneous task offloading with deep reinforcement learning in edge computing. In: Proceedings of 2022 8th IEEE International Conference on Cloud Computing and Intelligence Systems, CCIS 2022, pp. 558–564 (2022)
20. Zhu, B., et al.: Efficient offloading for minimizing task computation delay of NOMA-based multiaccess edge computing. IEEE Trans. Commun. **70**(5), 3186–3203 (2022)
21. Zhang, S., et al.: DRL-based partial offloading for maximizing sum computation rate of wireless powered mobile edge computing network. IEEE Trans. Wirel. Commun. **21**(12), 10934–10948 (2022)
22. Chen, Y., et al.: Dynamic task offloading for mobile edge computing with hybrid energy supply. Tsinghua Sci. Technol. **28**(3), 421–432 (2023)
23. Tong, Z., et al.: Stackelberg game-based task offloading and pricing with computing capacity constraint in mobile edge computing. J. Syst. Architect. **137**, 102847 (2023)
24. Seo, H., et al.: Differential pricing-based task offloading for delay-sensitive IoT applications in mobile edge computing system. IEEE Internet Things J. **9**(19), 19116–19131 (2022)
25. Wang, X., et al.: Decentralized scheduling and dynamic pricing for edge computing: a mean field game approach. IEEE/ACM Trans. Netw. **31**(3), 965–978 (2023)
26. Chen, S., et al.: Dynamic pricing for smart mobile edge computing: a reinforcement learning approach. IEEE Wirel. Commun. Lett. **10**(4), 700–704 (2021)
27. Chen, Y., et al.: Dynamic task offloading for internet of things in mobile edge computing via deep reinforcement learning. Int. J. Commun. Syst., e5154 (2022)
28. Xu, J., et al.: Online learning for offloading and Autoscaling in energy harvesting mobile edge computing. IEEE Trans. Cogn. Commun Netw. **3**(3), 361–373 (2017)
29. Qu, G., et al.: DMRO: a deep meta reinforcement learning-based task offloading framework for edge-cloud computing. IEEE Trans. Netw. Serv. Manage. **18**(3), 3448–3459 (2021)
30. Wang, J., et al.: Fast adaptive task offloading in edge computing based on meta reinforcement learning. IEEE Trans. Parallel Distrib. Syst. **32**(1), 242–253 (2021)
31. Yan, W., et al.: Survey on recent smart gateways for smart home: systems, technologies, and challenges. Trans. Emerg. Telecommun. Technol. **33**(6), e4067 (2022)

32. Dabin, J.A., et al.: A statistical ultra-wideband indoor channel model and the effects of antenna directivity on path loss and multipath propagation. IEEE J. Sel. Areas Commun. **24**(2), 752–758 (2006)
33. Cai, J., et al.: Deep reinforcement learning-based multitask hybrid computing offloading for multiaccess edge computing. Int. J. Intell. Syst. **37**(9), 6221–6243 (2022)
34. Wang, W., et al.: Trade-off analysis of fine-grained power gating methods for functional units in a CPU. In: Symposium on Low-Power and High-Speed Chips - Proceedings for 2012 IEEE COOL. Chips. XV. (2012)
35. Chen, E., et al.: SaaSC: toward pay-as-you-go mode for software service transactions based on blockchain's smart legal contracts. IEEE Trans. Serv., Comput. (2023)
36. Chargebee, what is pay as you go pricing model, on-line webpage (2022). https://www.chargebee.com/resources/glossaries/pay-as-you-go-pricing/
37. Zhao, N., et al.: Multi-agent deep reinforcement learning for task offloading in UAV-assisted mobile edge computing. IEEE Trans. Wirel. Commun. **21**(9), 6949–6960 (2022)
38. Van Hasselt, H., et al.: Deep reinforcement learning with double Q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, no. 1, pp. 2094–2100 (2016)
39. Lai, G., et al.: Modeling long- and short-term temporal patterns with deep neural networks. In: 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2018, pp. 95–104 (2018)
40. Liu, Z., et al.: Computation offloading and pricing in mobile edge computing based on Stackelberg game. Wirel. Netw. **27**(7), 4795–4806 (2021)
41. Li, F., et al.: Stackelberg game-based computation offloading in social and cognitive industrial internet of things. IEEE Trans. Industr. Inform. **16**(8), 5444–5455 (2020)
42. Liao, L., et al.: Online computation offloading with double reinforcement learning algorithm in mobile edge computing. J. Parallel Distrib. Comput. **171**, 28–39 (2023)