Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Distributed reinforcement learning-based memory allocation for edge-PLCs in industrial IoT

Tingting Fu[1], Yanjun Peng[1], Peng Liu[1], Haksrun Lao[2*] and Shaohua Wan[3]

Abstract

The exponential device growth in industrial Internet of things (IIoT) has a noticeable impact on the volume of data generated. Edge-cloud computing cooperation has been introduced to the IIoT to lessen the computational load on cloud servers and shorten the processing time for data. General programmable logic controllers (PLCs), which have been playing important roles in industrial control systems, start to gain the ability to process a large amount of industrial data and share the workload of cloud servers. This transforms them into edge-PLCs. However, the continuous influx of multiple types of concurrent production data streams against the limited capacity of built-in memory in PLCs brings a huge challenge. Therefore, the ability to reasonably allocate memory resources in edge-PLCs to ensure data utilization and real-time processing has become one of the core means of improving the efficiency of industrial processes. In this paper, to tackle dynamic changes in arrival data rate over time at each edge-PLC, we propose to optimize memory allocation with Q-learning distributedly. The simulation experiments verify that the method can effectively reduce the data loss probability while improving the system performance.

Keywords: Industrial internet of things, Edge-PLC, Resource allocation, Q-learning

Introduction

Internet of things (IoT) devices promote the intellectualization of industrial production and at the same time exponentially increase the amount of data to be handled and analyzed. The heavy burden on the central severs together with the transmission delay results in the inability to quickly and accurately obtain information from data. Edge computing appears to only share the workload placed on the central server and address the shortcoming [1, 2]. Multiple edge servers (ESes) [3] are common in real applications which distribute data computing tasks closer to data sources and users. In some complicated scenarios, such as the industrial Internet of things and smart city Internet of things, the hierarchical architecture and edge-cloud computing cooperation turn out to be very important and efficient [4, 5]. Thus, the industrial IoT supported by edge-cloud computing cooperation will potentially push traditional industries into a new stage of intelligence.

PLC is an electronic equipment used for digital operation in industrial production. It is in continuous progress to adapt to the complexity and uncertainty of modern industrial development. Modern PLCs gradually transform into edge-PLCs as a result of combining traditional logic control with networking, data collection, and processing [6, 7]. Most PLCs have work memory and load memory for storing user programs and data, respectively. Due to the cost control and ability of the equipment itself, the storage capacity of an edge-PLC is limited compared with a huge amount of production data, monitoring data, and user data. Possible data loss will affect the reliability of data analysis results.

*Correspondence: haksrunlao@hotmail.com

[2] Center of Engineering and Design, Chhong Cheng Chinese School, Phnom Penh, Cambodia
Full list of author information is available at the end of the article

This paper examines an intelligent factory's industrial control system using edge-PLCs. These edge-PLCs are embedded in factory equipment, receiving data streams from internal or external sensors, and putting the data into a pre-allocated memory space for analyzing and processing. IoT devices and edge-PLCs are the two main types of entities in the proposed scenario, as shown in Fig. 1. Edge-PLCs function as distributed intelligent agents that process data requests and produce control commands to avoid the high cost of central server [8].

Due to flexible manufacturing, various parameters of the industrial environment are likely to change more frequently over time. The characteristics of the data flow for each particular time period will differ even though the overall trend of the data flow in the industrial process may follow a particular type of distribution. In this case, the static memory allocation method for edge-PLCs is not able to achieve the optimal system performance, so it is necessary to study the dynamic allocation strategy of edge-PLC memory. We use a simple example to illustrate the importance of the dynamic memory allocation. As shown in Fig. 2, to simulate various real data flow in the manufacturing process, arrival data streams from Tp_1 to Tp_4 are all produced using a Poisson distribution, with Tp_1 having the maximum average arrival rate and Tp_4 having the minimum. It can be seen from the figure that the arrival data amount of Tp_1 is not always maximal in all time periods, nor is the arrival number of Tp_4 always minimal. When $T = 1$, a large amount of data indicated by Tp_1 arrives in the system, so it is necessary to allocate more memory space to Tp_1 to reduce data loss probability. However, if this allocation status is kept unchanged all the time, the system performance will decrease when $T = 5$. Because at this time, the amount of data indicated by Tp_2 increases significantly, while that of Tp_1 decreases. Therefore, the allocation scheme should be adjusted to give more memory space to Tp_2.

We need to analyze how to dynamically change the allocation instances of edge-PLC memory according to the amount of various types of data in the arriving data stream in each time period. The ultimate goal is to minimize the overall data loss probability of the system and optimize the resource utilization under constraints of limited memory capacity. To accomplish this, we model

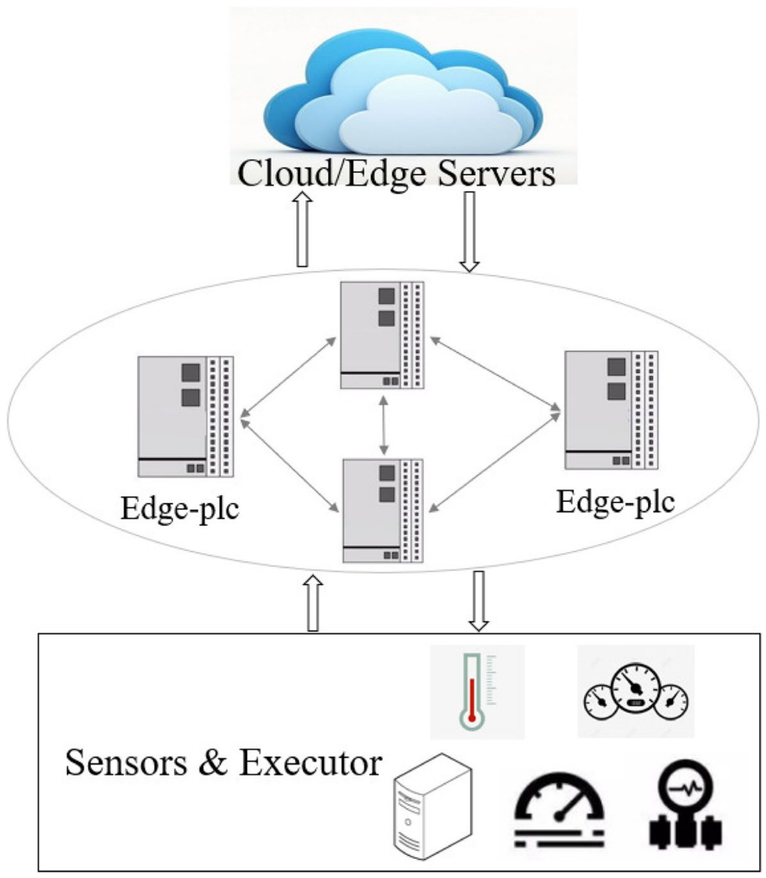

Fig. 1 Application scenario of edge-PLC

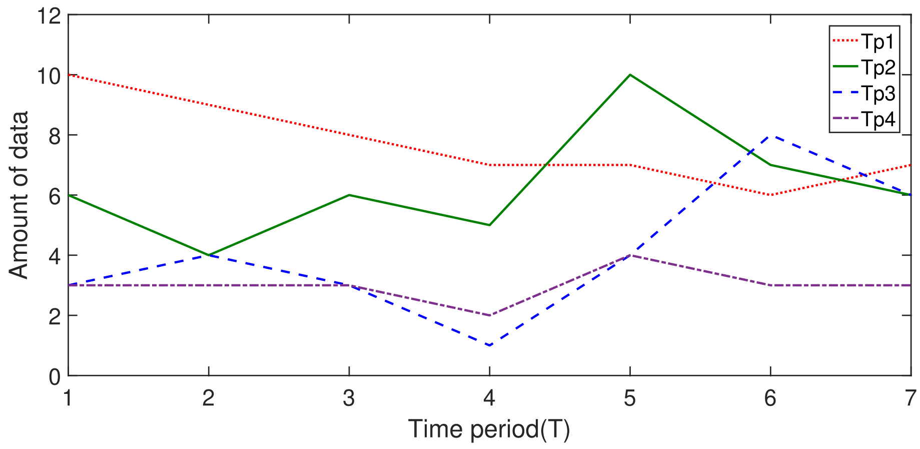

Fig. 2 The amount of various types of data arriving at different time periods

the issue as a Markov Decision Process (MDP) first. Then we apply reinforcement learning distributedly to interact with the environment and learn from experience to obtain the optimal strategy, and put it into practice. The main contributions of this paper are:

- The continuous time is divided into discrete variables and modeled as MDP to fit the variability of the data stream.
- Q-learning based dynamic allocation algorithm is designed with appropriate reward function to improve performance.
- The experiments are carried out on a practical application scenario, and prove that the algorithm can learn better allocation of each time period, to minimize the data loss probability while improving the utilization of resources.

The remainder of this paper is arranged as follows: Section II discusses approaches to solve resource allocation and scheduling problems and the advantages of reinforcement learning. Section III presents the mathematical models and the target problem. In Section IV, the detail of the dynamic allocation algorithm of edge-PLC memory, i.e., Q-learning based reinforcement learning, is explained. In Section V, the simulation experiment is carried out based on the synthetic data set. Finally, Section VI summarizes this paper.

Related Work

Many previous studies have used heuristic algorithms to solve the problem of resource management [9]. Chen et al. [10] proposed a distributed game theory approach to solve the computational migration problem. Wang et al. [11] proposed the alternative direction method to realize the optimization of cache resource allocation strategy in mobile edge calculation. Most of the researches on task scheduling and resource allocation focus on designing different and efficient static schemes [12]. Taking the fog computing system as an example, Liu et al. [13] applied queueing theory to study the delay, energy consumption, and payment cost in the offloading process. They proposed a multi-objective optimization problem with a minimized cost function and solved the problem by looking for offloading decisions. Wan et al. [14] propose the video segmentation algorithm based on the multi-modal linear features combination which divides the video sequence into segments of interests, and then extracts the video clips from these segments. A Lyapunov optimization-based energy-efficient offloading-decision algorithm is proposed in [15] to balance the energy-delay tradeoff based on various offloading-decision criteria. Considering the impact of different hop wireless communication ranges on task completion in a mobile edge computing scenario, the authors introduce the hop count k and select the neighboring nodes in the k-hop wireless communication range as the candidate edge servers [16]. Deng et al. [17] discussed the tradeoff between delay and energy consumption in a fog-cloud hybrid computing system and solved the related workload allocation problem accordingly. Dinh et al. [18] proposed an optimization framework for task offloading to optimize task allocation decisions and allocation of computing resources. This approach's drawback is that to optimize performance, the problem must be recalculated against the model and re-debugged in the experiment each time some aspects of the problem change. A multi-objective optimization with constraints that aims to maximize the acceptance rate and minimize the provider's cloud cost is how some research approaches resource allocation optimization in vehicular cloud computing [19]. However, the actual resource management problem becomes more and more complex. Firstly, the complexity of the environment leads to the difficulty of accurate modeling. In addition, the suggested method must perform well in different conditions given that the environment may be constantly changing.

Facing resource management problems, more and more studies suggest that reinforcement learning can be used to improve the method. DQN has been applied in task offloading and bandwidth allocation for multi-user mobile edge computing [20], which considers where to execute tasks and how much bandwidth should be allocated for dual communications. In [21], the authors take surrounding vehicles as a Resource Pool (RP). A distributed computation offloading strategy based on Deep Q-learning Network (DQN) is proposed to find the best offloading method to minimize the execution time of a compound task. Deng et al. [22] modeled the service resource allocation of the edge servers in edge computing as Markov Decision Process (MDP) and then used the reinforcement learning method to obtain the trained resource allocation strategy, which can always dynamically generate appropriate resource allocation scheme according to the system state. Value-function-based algorithms are widely used in reinforcement learning, these algorithms are more suitable for complex and changeable environments. Q-learning and SARSA are two classic algorithms. To address the issue of allocating radio network resources, Kaur et al. [23] proposed a multi-agent model-free reinforcement learning scheme and adopted Q-learning of reinforcement learning and SARSA decentralized cooperation to implement the resource allocation strategy under the presumption of enhancing energy efficiency and guaranteeing service quality. To maximize long-term advantages, Cui et al. [24] investigated the issue of dynamic resource allocation for multi-UAV communication networks. To simulate the dynamics and uncertainty in the environment, the long-term resource allocation problem is made into a random game problem with the maximum expected return, in which each UAV is an agent and each resource allocation scheme corresponds to the actions taken by the UAV, a multi-agent reinforcement learning framework is proposed. Each agent finds its own best strategy through learning according to its local observation, each agent independently executes a decision algorithm, but shares a common structure based on Q-learning. Baek et al. [25] aimed at the problem that a single fog node could not perform computationally intensive tasks in fog calculation, they proposed to use the deep reinforcement learning method to study the design of joint task offloading and resource allocation while ensuring user service quality. Their approach is to model the problem as a partially observable random game, using a deep recursive Q network approach to approximate the optimal value function to maximize the local reward for each fog node.

However, few references have targeted at the problem of PLC memory allocation for varying data flow in industrial Internet of things. Most existing work uses static allocation plans. Therefore, a light-weight and efficient dynamic allocation method is an emerging demand.

Problem Formulation and System Model

Description of dynamic resource allocation

Since there is no central server, each edge-PLC makes its decision distributedly. We consider that the total memory space of a single edge-PLC is divided into N memory blocks of the same size, and these memory blocks are allocated to m types of data. The allocated space cannot be greater than the total memory capacity of the PLC. Continuous time is divided into discrete variables $t = 0, 1, 2, ..., n$, and the duration of each time period is T. The real-time arrival rate of data in different time periods is disparate, so it is necessary to allocate appropriate memory space for the data that is about to arrive within each T. That is, the allocation of memory resources will change dynamically with the change of data arrival rate in different time periods, to achieve the effectiveness of adaptive resource allocation. As shown in Fig. 3, in an industrial scene where edge-PLCs are used, the memory partition of a single edge-PLC will change according to the data stream that will arrive in different time periods. The squares of four colors in the figure correspond to the four types of data in Fig. 2 respectively, and the size change of the squares represents the change of the memory space allocated to such data. m types of data in each T are likely to arrive randomly, and these data types share the memory of the PLC. We redistribute the memory space of the PLC for the data in the next time slice at every time t so that the data loss probability in each time slice is minimized, and then the global loss probability is also minimized.

We assume that each type of data is processed independently, and use $S_1, S_2, ..., S_m$ respectively represents the processing unit of each type of data. Each type of data can only be stored and processed by the corresponding type of memory space and processing unit, and the processing unit can process only one unit of data at a time. When data is waiting for processing in memory, it follows the queuing rule of first come, first served. When any data arrives, if the corresponding processing unit S_i is idle, it is assigned to S_i. If the same type of data is already present in the S_i, the system will try to store the data in the memory block if there is room for it. However, the input data is discarded if there is no room in the designated

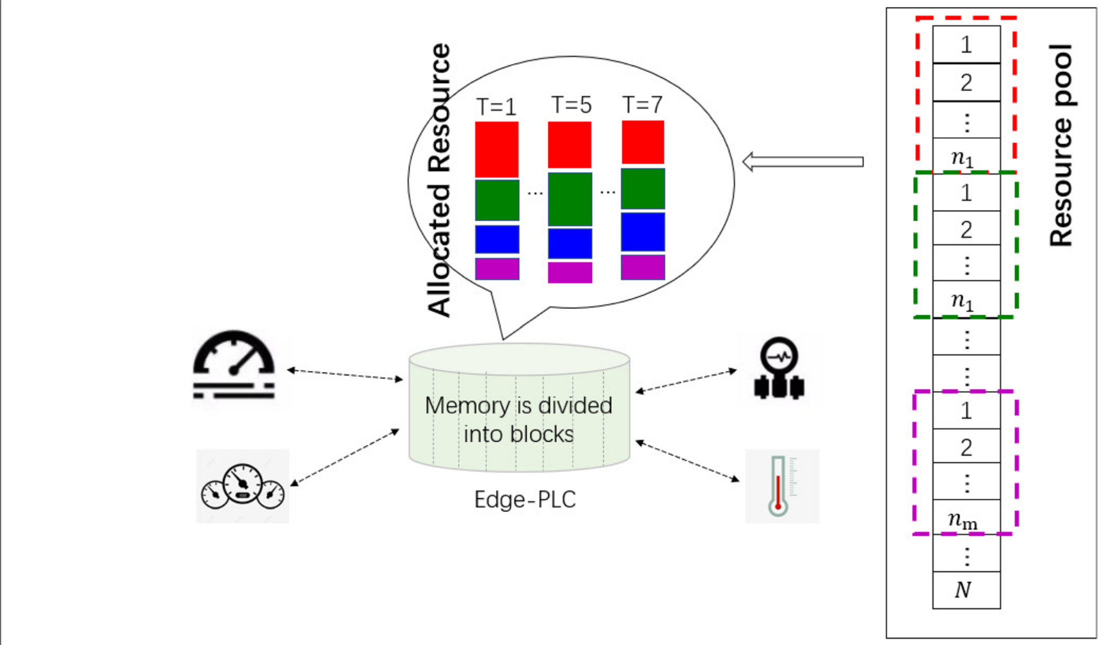

Fig. 3 System model of dynamic memory allocation of edge-PLCs

memory block. The memory occupied by the data is only freed after the data has been processed. Although the queue and processing unit of each type of data are independent of each other, the memory space of each type of data is influenced by one another because the whole PLC memory is shared. We regard the whole system as a complete environment, and all types of data are analyzed in the model. At each time *t*, a memory allocation strategy is chosen for the data arriving in the next time period. This strategy includes the memory space allocated to each type of data, and then the loss probability of each type of data and the overall data loss probability of the system are calculated according to the data volume, allocation method, and real-time processing rate of the PLC.

Vector $P_t = (n_t^1, n_t^2, ..., n_t^m)$ represents the resource allocation method at time *t*. In fact, the memory space allocated to Tp_i at time t is $l_t^i = n_t^i * x_i$. And $a_t (a_t \in A)$ represents the resource allocation strategy adopted at time *t*. And, A contains all possible memory partitioning method, that is, every item of A is a vector P_t, then the resource allocation in the next period $P_{t+1} = a_t$, where the definition of relevant variables is shown in Table 1.

Establish the MDP model

Data arrival and processing take place over several time periods, which requires careful consideration. The allocation strategy chosen at the last minute determines the resource allocation plan for each time period. MDP can be used to describe this process, and Table 1 provides

Table 1 Symbolic variables used by the system model

Variable	Meaning
Tp_i	Data type *i*
l_t^i	Absolute memory capacity allocated to Tp_i at time *t*
x_i	The actual size of Tp_i type data
n_t^i	The allocated data unit quantity for Tp_i at time *t*
m	The total number of data types that exist in the system
P_t	The partition of memory at time *t*.
S	The set of all the states
A	The set of all the actions
R_{t+1}	The reward value corresponding to (s_t, a_t)
Mem	Edge PLC memory maximum capacity
$Ploss_t^i$	Loss probability of Tp_i between t and $t + 1$
Ar_t^i	The amount of Tp_i that arrives between t and $t + 1$
$loss_t^i$	The amount of Tp_i that is lost between t and $t + 1$

definitions of key variables that are used. To be in accordance with other work adopting reinforcement learning, we regard each edge-PLC as an intelligent agent.

(1) State space

$s_t = (t, P_t)$ represents the state of the system at the beginning of time *t*. In the same industrial production process, the data flow generated by the equipment during the same time period across days is similar. In the learning phase, the agent learns the most appropriate memory allocation method for each time period of the day, that is, which allocation method should be adopted at each time *t*. The results of this learning can then be used to guide the rational allocation of resources in subsequent industrial production. The action taken at the current moment determines the allocation method in the next time slice, which in turn determines the amount of data still waiting in the queue for processing at the next moment. These data should also be considered when selecting the action at the next moment, so the state at the current moment will affect the state and action at the next moment.

(2) Action space

$a_t(a_t \in A)$ is used to represent the actions taken by the agent under the state s_t. The allocation of memory space is constrained due to the limited number of data types and available memory. All allocation methods can be found as action space. The action space includes all possible combinations of the amount of various types of data that can be stored simultaneously in the memory of the PLC. We can also eliminate some allocation methods that are known in advance to have poor performance, such as having too little memory allocated to each type of data.

(3) Reward function

At the end of each time slice, the data loss probability can be calculated, and the reward function can be defined as: if the agent takes an action and the data loss probability for that time period is 0, the reward value for that action in the current state is positive. At this point, the reward value should be inversely proportional to the allocated memory space capacity. The less memory space occupied, the greater the reward value is. The reward value is the remaining memory space that has not been allocated to any kind of data, (s_t, a_t) corresponds to a reward value of

$$R_{t+1} = Mem - \sum_{i=1}^{m} l_t^i \tag{1}$$

The reward value is negative if the probability of data loss is greater than 0. The reward value is inversely proportional to the sum of the loss probability of all types of data in each time period. The larger the loss probability is, the smaller the reward value is. Here, the negative value of the sum of the loss probability of all types of data is taken as the reward value. In this case, the reward value corresponding to (s_t, a_t) is

$$R_{t+1} = -\sum_{i=1}^{m} Ploss_t^i \tag{2}$$

$$Ploss_t^i = \frac{loss_t^i}{Ar_t^i} \tag{3}$$

A sequence $seq = [s_t, a_t, s_{t+1}, a_{t+1}, ...]$ at each moment t can be obtained according to the state s_t and strategy π at the current moment. The cumulative return of this sequence can be expressed as

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4}$$

The agent should consider both the immediate and long-term rewards when calculating the total return, which is why γ is the discount factor. However, the longer the interval is, the more inaccurate the future reward value will be. Therefore, discount factors should be used to reduce the proportion of future rewards in current returns.

Because strategy π is a probability distribution, there may be a variety of different *seq*, using the expectation of G_t to evaluate the cumulative return of state $s_t = s$, as shown in (5):

$$Q_\pi(s,a) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t = s, a_t = a] \tag{5}$$

Given a strategy β, ρ_β represents the probability density function of β, and the expected cumulative return of strategy π is

$$\begin{aligned} U_\beta(\pi) &= \int_{x \in S} \int_{y \in A} \rho_\beta(x) Q_\pi(x,y) dx dy \\ &= E_{x \sim \rho_\beta, y \sim \beta}[Q_\pi(x,y)] \end{aligned} \tag{6}$$

The ultimate goal is to maximize the cumulative return expectation, so the problem turns into finding the strategy π^* in (7):

$$\pi^* = \arg\max_\pi U_\beta(\pi) \tag{7}$$

Proposed solutions

Reinforcement learning

Reinforcement learning can solve problems in a complex environment where the model is unknown, evaluate the value of the current behavior through interaction with the environment, and then improve the next behavior to achieve the final goal. In the scenario we proposed, the agent can continuously improve its behavior by evaluating performance indicators such as data loss probability and resource utilization, to achieve better allocation of edge-PLC memory resources. And reinforcement learning is suitable for the analysis of time series data, the current action will have an impact on subsequent results. Because of this, this paper analyzes the allocation of memory resources using reinforcement learning to discover the best strategy through interaction with the environment and implements the adaptive dynamic allocation strategy. This is done to effectively deal with the change in data arrival rate in different time periods.

Reinforcement learning algorithms include model-based methods and model-free methods. The model-based method must know the state transition probability, but in the resource allocation system, due to the randomness and uncertainty of the process, the accurate state transition probability matrix cannot be obtained.

The proposed Q-learning based algorithm

Q-learning is an off-policy learning algorithm in reinforcement learning. It can explore more states than SARSA and its behavior strategy is incompatible with the target strategy. In the scene studied in this paper, there are many optional actions, and the results of different actions are quite different. To make the algorithm find the global optimal solution as much as possible, and also learn some "unfavorable" actions to the target in the learning stage, to prevent the subsequent selection of these actions, Q-learning is more suitable to solve the problems proposed in this paper. The model-free method of Q-learning is considered to solve the problem. The reasons for using the Q-learning method are as follows:

(1) Q-learning algorithm has good flexibility and is self-adaptive. It can dynamically adjust the memory resource allocation scheme according to the arrival rates of data flows at different time periods.
(2) In the Monte-Carlo method of reinforcement learning, the experience update can only be carried out after the complete resource allocation process from the initial state to the termination state is completed each time. If the complete resource allocation process takes a long time, there will be a large update delay. Temporal-Difference methods can be updated immediately after a time step, so it has a wider application range, and Q-learning is one of Temporal-Difference methods.
(3) Q-learning is an algorithm based on a value function, which only needs to describe and solve problems through the state-action pair of parameters, while the strategy-based algorithm needs to be trained with the same strategy to obtain an appropriate model. However, in the scene proposed in this paper, uncertainty and volatility are large, so the strategy-based iterative method is not applicable.

The updated formula of the Q-learning algorithm's state-action value function (Q value) is

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)] \quad (8)$$

Where $\alpha(\alpha \in [0,1])$ is the learning rate. This value represents the extent to which the old value is covered in the learning process. The larger α is, the less influence the previous learning experience has on the subsequent decision.

The process of the Q-learning algorithm is shown in Fig. 4, where the $\varepsilon - greedy$ strategy is given, $\varepsilon(0 < \varepsilon < 1)$, and a number between 0 and 1 will be randomly generated in the algorithm before the agent makes a decision each time. If this random number is less than ε, the agent will select the action with the largest Q value corresponding to the current state according to the accumulated experience; otherwise, the agent will randomly select an action in the action space for execution. In other words, the agent has the probability of ε to make decisions based on experience, and the probability of $1 - \varepsilon$ to randomly select actions to explore new optimal solutions beyond the current experience. Before the agent starts the learning process, the Q table is first initialized, then in each episode, the algorithm starts from the initial state. At each state, the agent selects an action based on the $\varepsilon - greedy$ strategy, obtains the reward value of the state-action pair, and the environment transitions to the next state, and then the Q value is updated according to (8). After the end of this state, the next state is regarded as the current state, and the operation of "action selection" is repeated until the current state is terminated, then this episode terminates. The above operation is repeated in the next episode until all Q values converge or the iteration termination condition is reached.

Based on Q-learning, we propose Dynamic Memory Resource Allocation (DMRA) algorithm for edge-PLCs. DMRA is located in the edge server which is close to edge-PLCs because it requires large computation power and sensitive to delay. Its process is shown in algorithm 1. Firstly, the Q value table is initialized, because this Q value table is updated by the agent in the process of continuous learning and accumulation of experience, the main purpose of initialization is to determine the action space in the table. And in the subsequent process, the Q value of each state under all actions is written into the table. Each iteration starts from the initial state. At each time t, the agent first determines whether the current state has been stored in the Q table, if not, it adds the state to the table. Then, the $\varepsilon - greedy$ strategy is adopted to select the allocation method at the current moment. The reward value of the action is calculated according to (1) and (2). The state at the next moment is obtained, then the Q value is updated. After that, the program enters the next moment. Once the time is up, a complete

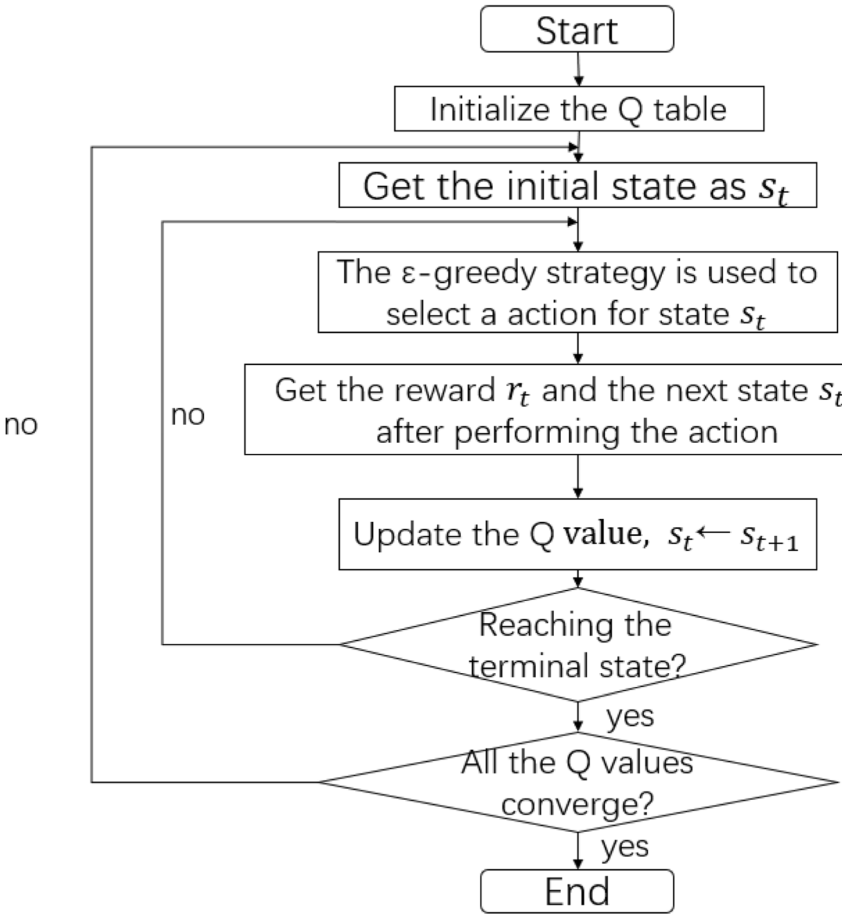
Start
Initialize the Q table
Get the initial state as s_t
The ε-greedy strategy is used to select a action for state s_t
Get the reward r_t and the next state s_{t+1} after performing the action
Update the Q value, $s_t \leftarrow s_{t+1}$
Reaching the terminal state?
yes
All the Q values converge?
yes
no
no
End

Fig. 4 Q-learning algorithm flow chart

iteration process is completed, and the next iteration will continue. When the termination condition of the iteration is reached, the program will be terminated.

```
Input:
    $A, Mem, \alpha, \gamma, \varepsilon$
Output:
    Q value table
1:  Initializes the Q value table with $A$;
2:  for each episode do
3:      Initialize the system state $s_t$;
4:      for each time period $t$ do
5:          if $s_t$ is not in the Q value table then
6:              Add $s_t$ to the Q value table;
7:          end if
8:          Select an action $a_t$ based on the $\varepsilon - greedy$ strategy;
9:          Get reward $R_{t+1}$ with (1) and (2);
10:         Get new state $s_{t+1}$
11:         Update the Q value table with (8);
12:         $s_t \leftarrow s_{t+1}, a_t \leftarrow a_{t+1}$;
13:     end for
14: end for
15: return Q value table
```

Algorithm 1 Dynamic Memory Resource Allocation for edge-PLC (DMRA)

Experiments

Experimental setup

In order to provide more realistic performance evaluations, we fully account for reality in the experiment and simulate the arrival data flow as well as the PLC's processing power for different types of data. Production data flows are generated in the same industrial environment for ten days. In a real scenario, IoT devices connected to different servers are almost the same kind. And in the same industrial production process, the difference in the data flow rates generated by the devices at the same time every day will be relatively small. To make the data as close to reality as possible, the data set refers to the transmission data size of the four types of industrial equipment collected in the literature [6], as shown in Table 2. The synthetic data set is simulated by the above conditions. To better show the experimental results, this experiment considers the allocation of the edge-PLC memory capacity of 10kB and uses one hour as a time slice to study the memory allocation strategy for each day. The average arrival

Table 2 Four types of parameters of factory equipment

	Device type	Device name	Amount of data
1	Heat exchanger	Heat-HR-A3-V52	8B
2	Smart meters	Elemeter-LC-EX8016	18B
3	Door system	door-DGM-V52	20B
4	Power supply	Power-BYD-V52	24B

rate of data in each time slice of a day in the simulation data set is shown in Fig. 5, where the data on device types 1 to 4 are $Tp_1 \sim Tp_4$. And the learning rate and discount factor of the DMRA algorithm are set to 0.01 and 0.9, respectively. The performance is compared with two baseline methods, i.e., greedy and Stackelberg game-based [26].

Simulation results and analysis

(1) The changing trend of the total loss probability of all data flows during the reinforcement learning training process

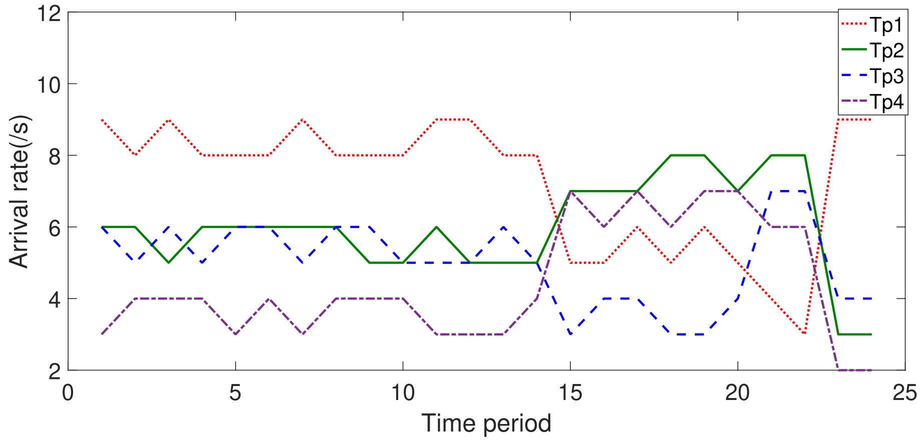

Fig. 5 The average rate of arrival in each time period

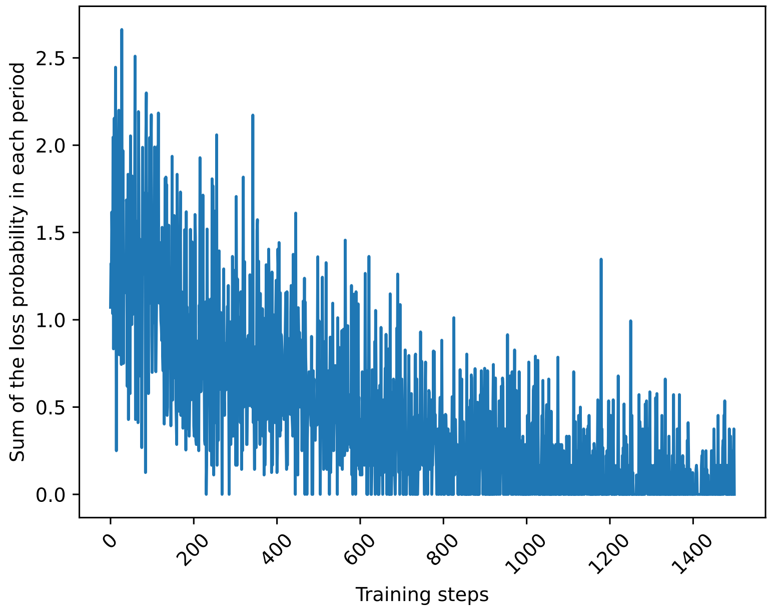

Fig. 6 The total data loss probability of each period in the training process

Figure 6 shows the sum of all types of data loss probability in each iteration of the DMRA algorithm. It can be seen that after a certain number of iterations, the agent gradually finds the optimal solution. The algorithm uses the $\varepsilon - greedy$ strategy, the agent may randomly take actions every time, so the result is an oscillating curve. However, because each decision made by the agent will have an impact on the corresponding Q value, the experiment's ultimate aim is to develop a Q value table that will be used to direct future memory allocation. In this experiment, to speed up the convergence rate of the Q-value table, some cases where too little memory space is allocated to each type of data are removed when initializing the action space, the performance of these allocation methods will be significantly poorer.

(2) The proportion of the amount of data arriving in each time period and the proportion of the amount of data that can be stored in the allocated space

After the DMRA algorithm trains the Q table, a new piece of data simulated under the same conditions is used to test the result of memory allocation based on the Q table. Figure 7(a) shows the proportion of various types of data in the data stream that arrive in each time period of the day. We can see that Tp_1 had the most of the amount of data in the first 14 hours, followed closely by Tp_2 and Tp_3, and Tp_4 had the least. But, the proportions of Tp_1 and Tp_3 decrease from the 15th to the 20th hour, while the proportions of Tp_2 and Tp_4 show an increase. In the 21st and 22nd hours, the proportion of Tp_3 increases significantly. In the last two hours, the proportion of Tp_1 becomes the largest.

Let this new data select the action with the largest Q value corresponding to the current state in the Q table at each time period. Figure 7(b) displays the percentage of the various data types that can be stored after the memory is divided in each time period. As can be seen from the figure, the changing trend of the memory partitioning method in some time periods is not exactly the same as the changing trend of the arriving data stream. This is because memory resource allocation is not only affected by the current time period. The data waiting for processing in the queue in the previous time period, may still be waiting in the current time period. So these data will still occupy memory space and keep waiting. The data that is currently in the queue must also be taken into account when allocating memory resources. As a result, the allocation of memory resources depends on the amount of data and the allocation strategy used in the previous and the current time period. Even if the data stream is the same in some time periods, the memory division method may be different.

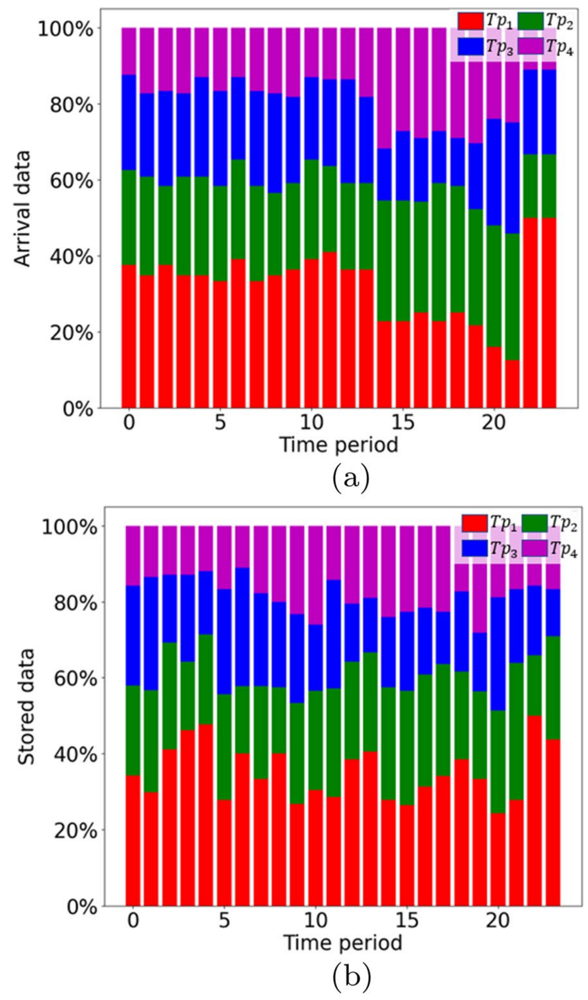

Fig. 7 **a** The proportion of the amount of each type of data that arrives. **b** The proportion of the amount of different types of data that can be stored in the allocated memory space

(3) Comparison of loss probability of the same data stream in random strategy, greedy algorithm, and DMRA algorithm

Figure 8(a) and (b) respectively compare the data loss probability of the actions selected by the agent according to the random strategy, greedy algorithm, and DMRA algorithm at each time period and the total loss probability of all time periods. It can be seen that the loss probability caused by the random strategy and the greedy algorithm is higher than that caused by the DMRA algorithm at each time period. The combined data loss probability of these two approaches is significantly higher than that of the DMRA algorithm, indicating that the proposed approach can successfully achieve the goal of lowering data loss probability. In the majority of time periods, the DMRA algorithm forces the agent to choose the action

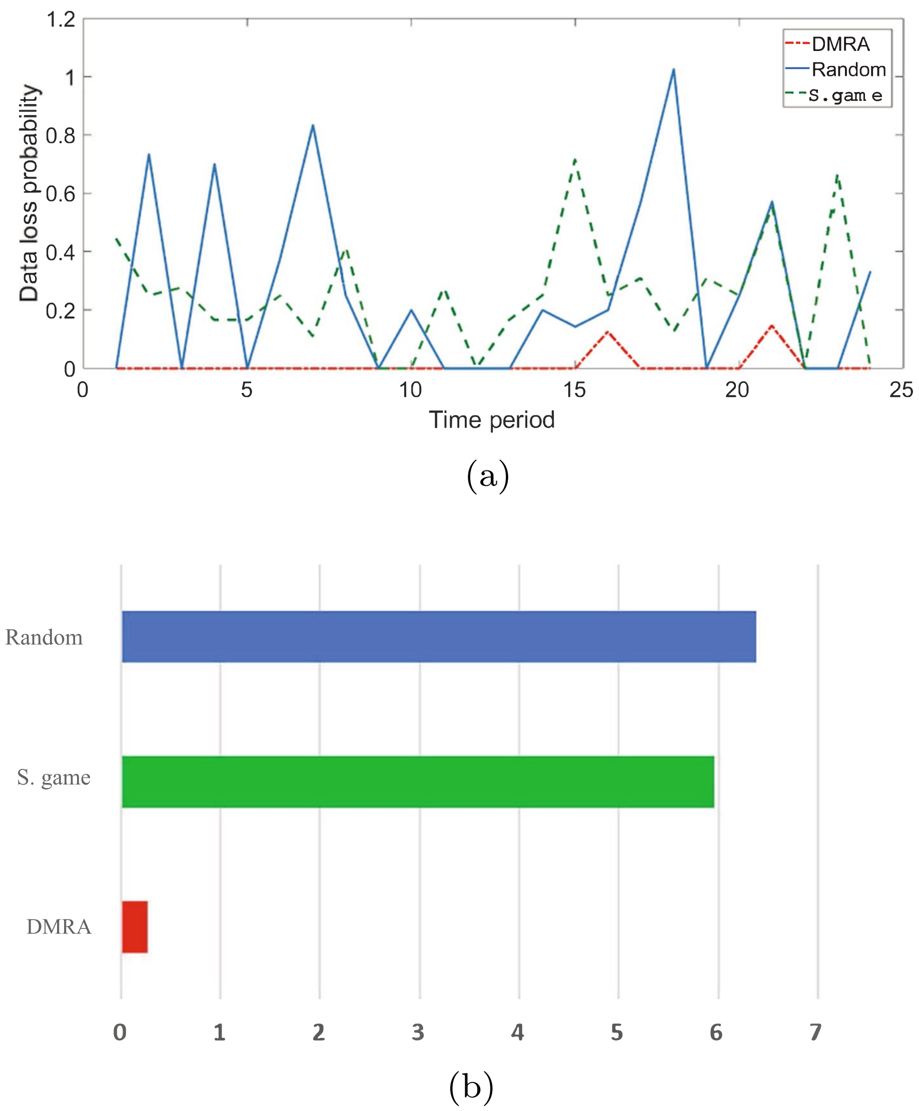

Fig. 8 **a** The loss probability of each time period. **b** Total loss probability for all time periods

with the lowest loss probability, but there are a few when the chosen actions still result in partial data loss. This may be because the data arrival rate is too high relative to the data processing rate in certain time periods. At this time, the reason for the data loss is not only the size of the allocated memory space, but the more important factor is the PLC processing capability does not match the data arrival rate. Figure 9 shows the sum of the data loss probability for all time periods of Tp_1 at different average processing rates when the agent selects actions for the same data stream according to the same Q table. It is the average value after the experiment has been repeated for 50 times. When the processing rate is small, even if the strategy is selected in the same way, it will cause a larger loss probability, which shows that the loss probability will be affected by the data processing rate.

(4) Performance comparison between static allocation and dynamic allocation of memory resources

Figure 10 compares the total memory space occupied by the actions selected by the dynamic memory resource allocation method and the space occupied by the optimal static allocation method calculated by the Markov chain method described in our previous work [7] at each time period for processing the same piece of data. Because static allocation uses an unchanged allocation method, the amount of memory used over time is constant. Among

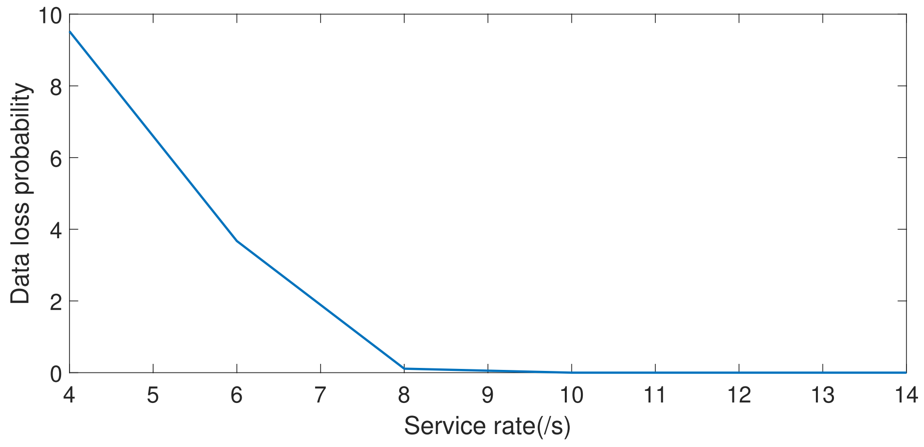

Fig. 9 Loss probability at different average processing rates

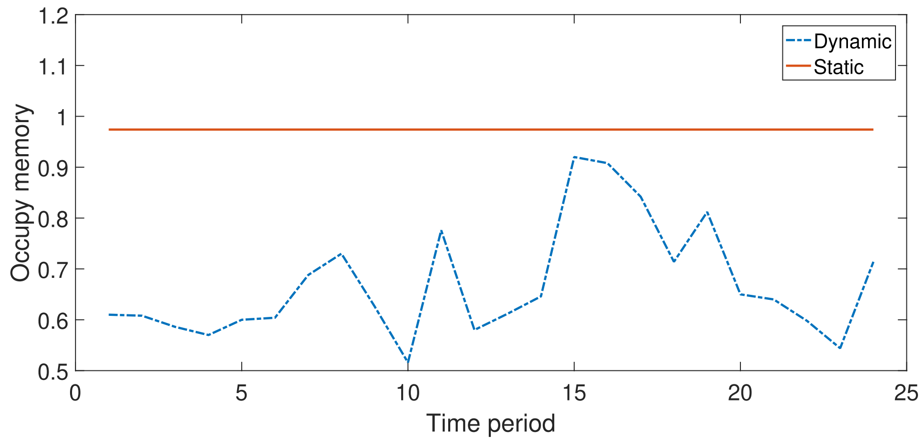

Fig. 10 Static allocation and dynamic allocation occupy memory space size

them, the memory space used by dynamic allocation in each time period is significantly lower than that of static allocation, this is because the reward function is designed so that the action taking up less space is rewarded more for the same loss probability. In static allocation, the memory resource allocation needed for the time period with the most data is typically chosen as the final result to ensure that the loss probability of each time period is the smallest. However, during other time periods, this allocation will have more unused space for the data stream, which lowers the utilization of the resources assigned.

The experiment utilizes simulation data from the same day to test the program's running time. As shown in Fig. 11, the DMRA algorithm trains the Q table and makes a strategy for the data of that day and is compared with the execution of the Markov chain method [7], the execution time is reduced by about half. Static allocation

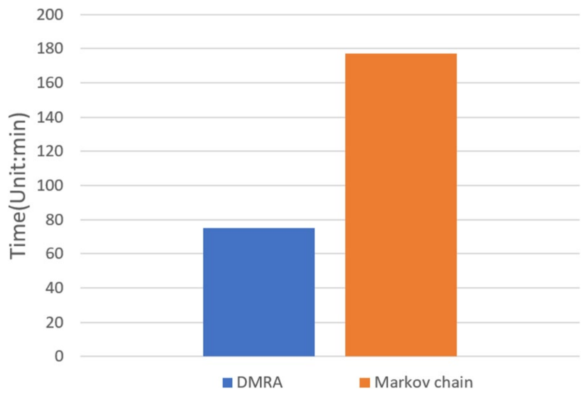

Fig. 11 Comparison of running time between Markov Chain and DMRA algorithm

takes longer to run because it has to compare every memory resource allocation approach before settling on the best one. Therefore, even the static allocation method can find the allocation method that minimizes the loss probability, and the advantage of the dynamic allocation proposed in this paper is also greater.

Conclusion

In the memory resource allocation problem of the edge-PLC, because the various types of data streams generated by the industrial production system in different time periods are also different, if the allocation method is static, it may not achieve good performance. Therefore, we divide continuous time into discrete variables, analyze the arrival data stream of each time period, and consider changing the method of memory allocation with the change of arrival rates of data flow of different time periods to realize the adaptive dynamic allocation of memory resources. We model the problem as an MDP and design a dynamic allocation algorithm of memory resources based on the Q-learning method of reinforcement learning to learn the most efficient allocation method for each time period. Reinforcement learning algorithms can interact with dynamic environments and learn experiences, so even in different scenarios, this method can learn the best resource allocation method in that scenario. Simulation experiments show that the dynamic allocation algorithm of memory resources can effectively improve system performance.

Abbreviations
Iiot: Industrial internet of things; PLC: Programmable logic controller.

Acknowledgements
The authors thank the editor and anonymous reviewers for their helpful comments and valuable suggestions.

Authors' contributions
Tingting Fu found the target problem and proposed the solution. Yanjun Peng completed most of the writing of this manuscript. Peng Liu led the write up of the manuscript. Haksrun Lao took part in the discussion of the solution and the writing. Shaohua Wan helped in revising the paper and gave many useful suggestions. All authors have read and approved the manuscript.

Authors' information
Tingting Fu received the B.S. degree from Hangzhou Dianzi University, Hangzhou, China, in 2000, and the M.S. degree from Zhejiang University, Hangzhou, in 2005. She is currently an Associate Professor at Hangzhou Dianzi University. She was a Visiting Scholar at Lehigh University, Bethlehem, PA, USA, from 2014 to 2015. Her research interests are in the area of Internet of Things, big data, wireless networks, and mobile computing.
Yanjun Peng was a graduate student at Hangzhou Dianzi University. Her research interest is industrial Internet of things and edge computing. Peng Liu received his B.S. and M.S. in Computer Science and Technology from Hangzhou Dianzi University respectively in 2001 and 2004, and Ph.D in Computer Science and Technology from Zhejiang University in 2007, China. Currently, he is an associate professor at Hangzhou Dianzi University. His research interests include Internet of things, edge computing, and vehicular ad-hoc networks.
Haksrun Lao is a teacher at Chhong Cheng Chinese School, Cambodia. His research interests include Internet of things and visual light communication. Shaohua Wan is a professor at Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China. His research interests include edge computing, Internet of vehicles and machine learning.

Funding
This work is supported by the Natural Science Foundation of China under Grant 62172134.

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. [2]Center of Engineering and Design, Chhong Cheng Chinese School, Phnom Penh, Cambodia. [3]Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, Shenzhen, China.

Received: 6 August 2022 Accepted: 11 October 2022
Published online: 28 October 2022

References
1. Wu H, Li X, Deng Y (2020) Deep learning-driven wireless communication for edge-cloud computing: opportunities and challenges. J Cloud Comput 9:21
2. Chen C, Li H, Li H, Fu R, Liu Y, Wan S (2022) Efficiency and fairness oriented dynamic task offloading in internet of vehicles. IEEE Trans Green Commun Netw 1. https://doi.org/10.1109/TGCN.2022.3167643
3. You Q, Tang B (2021) Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. J Cloud Comput 10(1):41
4. Wu H, Zhang Z, Guan C, Wolter K, Xu M (2020) Collaborate edge and cloud computing with distributed deep learning for smart city internet of things. IEEE Internet Things J 7(9):8099–8110. https://doi.org/10.1109/JIOT.2020.2996784
5. Zhang Z, Wang N, Wu H, Tang C, Li R (2021) Mr-dro: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments. IEEE Internet Things J 1–1. https://doi.org/10.1109/JIOT.2021.3126101
6. Wu H, Yan Y, Sun D, Wu H, Liu P (2021) Multi buffers multi objects optimal matching scheme for edge devices in iiot. IEEE Internet Things J 8(14):11514–11525. https://doi.org/10.1109/JIOT.2021.3053017
7. Peng Y, Liu P, Fu T (2020) Performance analysis of edge-plcs enabled industrial internet of things. Peer Peer Netw Appl 13(5):1830–1838
8. Safavat S, Sapavath NN, Rawat DB (2020) Recent advances in mobile edge computing and content caching. Digit Commun Netw 6(2):189–194. https://doi.org/10.1016/j.dcan.2019.08.004
9. Chen J, Du T, Xiao G (2021) A multi-objective optimization for resource allocation of emergent demands in cloud computing. J Cloud Comput 10:17
10. Chen X, Jiao L, Li W, Fu X (2016) Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans Networking 24(5):2795–2808. https://doi.org/10.1109/TNET.2015.2487344
11. Wang C, Liang C, Yu FR, Chen Q, Tang L (2017) Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. IEEE Trans Wirel Commun 16(8):4924–4938. https://doi.org/10.1109/TWC.2017.2703901
12. Sadatdiynov K, Cui L, Zhang L, Huang JZ, Salloum S, Mahmud MS (2022) A review of optimization methods for computation offloading in edge computing networks. Digit Commun Netw. https://doi.org/10.1016/j.dcan.2022.03.003

13. Liu L, Chang Z, Guo X, Mao S, Ristaniemi T (2017) Multiobjective optimization for computation offloading in fog computing. IEEE Internet Things J 5(1):283–294
14. Wan S, Ding S, Chen C (2022) Edge computing enabled video segmentation for real-time traffic monitoring in internet of vehicles. Pattern Recog 121:108146. https://doi.org/10.1016/j.patcog.2021.108146
15. Wu H, Sun Y, Wolter K (2020) Energy-efficient decision making for mobile cloud offloading. IEEE Trans Cloud Comput 8(2):570–584. https://doi.org/10.1109/TCC.2018.2789446
16. Chen C, Zeng Y, Li H, Liu Y, Wan S (2022) A multi-hop task offloading decision model in mec-enabled internet of vehicles. IEEE Internet Things J 1. https://doi.org/10.1109/JIOT.2022.3143529
17. Deng R, Lu R, Lai C, Luan TH, Liang H (2016) Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet Things J 3(6):1171–1181
18. Dinh TQ, Tang J, La QD, Quek TQ (2017) Offloading in mobile edge computing: Task allocation and computational frequency scaling. IEEE Trans Commun 65(8):3571–3584
19. Wei W, Yang R, Gu H, Zhao W, Chen C, Wan S (2021) Multi-objective optimization for resource allocation in vehicular cloud computing networks. IEEE Trans Intell Transp Syst 1–10. https://doi.org/10.1109/TITS.2021.3091321
20. Huang L, Feng X, Zhang C, Qian L, Wu Y (2019) Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. Digit Commun Netw 5(1):10–17. https://doi.org/10.1016/j.dcan.2018.10.003
21. Chen C, Zhang Y, Wang Z, Wan S, Pei Q (2021) Distributed computation offloading method based on deep reinforcement learning in icv. Applied Soft Computing 103:107108. https://doi.org/10.1016/j.asoc.2021.107108
22. Deng S, Xiang Z, Zhao P, Taheri J, Gao H, Yin J, Zomaya AY (2020) Dynamical resource allocation in edge for trustable Internet-of-things systems: A reinforcement learning method. IEEE Trans Ind Inform 16(9):6103–6113. https://doi.org/10.1109/TII.2020.2974875
23. Kaur A, Kumar K (2020) Energy-efficient resource allocation in cognitive radio networks under cooperative multi-agent model-free reinforcement learning schemes. IEEE Trans Netw Serv Manag 17(3):1337–1348. https://doi.org/10.1109/TNSM.2020.3000274
24. Cui J, Liu Y, Nallanathan A (2020) Multi-agent reinforcement learning-based resource allocation for uav networks. IEEE Trans Wirel Commun 19(2):729–743. https://doi.org/10.1109/TWC.2019.2935201
25. Baek J, Kaddoum G (2021) Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks. IEEE Internet Things J 8(2):1041–1056. https://doi.org/10.1109/JIOT.2020.3009540
26. Li Q, Lu C, Cao B, Zhang Q (2019) Caching resource management of mobile edge network based on stackelberg game. Digital Communications and Networks 5(1):18–23. https://doi.org/10.1016/j.dcan.2018.10.006

Publisher's Note