Deep Reinforcement Learning-Based Edge Computing Offloading Algorithm for Software-Defined IoT

Xiaojuan Zhu
Anhui University of Science and Technology
Tianhao Zhang (✉ iamzth@126.com)
Anhui University of Science and Technology
Jinwei Zhang
Anhui University of Science and Technology
Bao Zhao
Anhui University of Science and Technology
Shunxiang Zhang
Anhui University of Science and Technology
Cai Wu
Anhui University of Science and Technology

Research Article

Keywords: Edge computing, Computing offloading, Software defined network, Internet of things, Deep reinforcement learning

Posted Date: October 26th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2150294/v1

Deep Reinforcement Learning-Based Edge Computing Offloading Algorithm for Software-Defined IoT

Xiaojuan Zhu, Tianhao Zhang, Jinwei Zhang, Bao Zhao, Shunxiang Zhang and Cai Wu

School of Computer Science and Engineering, Anhui University of Science and Technology, No.168, Taifeng Street, Huainan, 232001, Anhui, China.

*Corresponding author(s). E-mail(s): iamzth@126.com; Contributing authors: xjzhu@aust.edu.cn; jwzhang@aust.edu.cn; bzhao@aust.edu.cn; sxzhang@aust.edu.cn; caiw2020@163.com;

Abstract

With the amount of data generated by Internet of Things (IoT) devices increase dramatically, the insufficient computing ability of terminal devices becomes obvious when processing massive computing tasks. The computing tasks need to be offloaded from resource-constrained devices to edge servers with stronger computing capability. It is a challenge for computing offloading to achieve global optimization with multiple objectives such as minimizing task completion times, optimizing energy consumption and maintaining load balancing as the network state and task demands dynamically change. This paper presents optimized edge computing offloading algorithm for software-defined IoT. First, to provide global state for making decisions, a software defined edge computing (SDEC) architecture is proposed. The edge layer is integrated into the control layer of software-defined IoT, and multiple controllers share the global network state information via east-west message exchange. Moreover, an edge computing offloading algorithm in software-defined IoT (ECO-SDIoT) based on deep reinforcement learning is proposed. It enables the controllers to offload the computing task to the most appropriate edge server according to the global states, task requirements, and reward. Finally, the performance metrics for edge computing offloading were evaluated in terms of unit task processing latency, load balancing of edge servers, task processing energy consumption, and task completion rate, respectively. Simulation results show that ECO-SDIoT can effectively reduce task completion time and energy consumption compared with other strategies.

Keywords: Edge computing, Computing offloading, Software defined network, Internet of things, Deep reinforcement learning

1 Introduction

The amount of data generated by Internet of Things (IoT) devices has increased dramatically in recent years. Usually, these IoT devices have limited computational ability and therefore need to offload computational tasks from the resource-constrained device to the edge servers with more computational power to meet the needs of low latency and bandwidth saving[1]. However, the computational resources of edge servers are equally limited compared to cloud servers. As more and more data needs to be analyzed, processed and stored on the edge servers, computing tasks need to be offloaded to the appropriate edge servers for fast processing of large volumes of tasks. Typically, edge computing applications can be divided into multiple subtasks[2]. When the local edge server lacks sufficient computing power, the divided subtasks must be offloaded to a different edge server for execution, and the results must be returned after execution.

Previous edge computing offloading approaches either consider only minimizing latency, ignoring the energy consumption of the offloading[3], or only minimizing energy consumption, ignoring the time constraints that the computing task can tolerate[4]. Some researchers have proposed edge computing offloading methods that minimize the system cost, but they only acquire computational resources from other edge servers through local message requests, and cannot offload flexibly and intelligently from a global perspective[5, 6].

It is a challenge for computing offloading to achieve global optimization with multiple objectives such as minimizing task completion times, optimizing energy consumption and maintaining load balancing as the network state and task demands dynamically change. When the network and task change rapidly, it is difficult for the heuristic algorithm to obtain or accurately predict the comprehensive statistics, the offloading performance may degrade, and it is difficult to make adaptive adjustments. The proposed intelligent computing offloading algorithms[7–9] still have much room for improvement in environment perception, task modeling and parameter optimization.

In order to enable computing offloading to be handled in an adaptive and intelligent manner, this paper achieves efficient offloading based on a software-defined IoT architecture. Software Define Network (SDN) architecture enables the separation of hardware and software through virtualization and abstraction, with the benefit of being able to define and extend the functionality of the entire system in a flexible manner[10]. Software-defined edge computing in the IoT environment decouples upper layer IoT applications from the underlying physical resources and builds dynamically reconfigurable intelligent edge services[11]. The advantage of this new architecture lies in the logical centralized control of distributed network nodes. SDN architecture allows for a global view and flexible reprograming. SDN-based traffic engineering[12] and routing optimization[13], and other aspects have received increased attention. However, there has been little research on edge computing offloading under the SD-IoT architecture. To the best of our knowledge, however, the problem of edge computing offloading in Software-Defined IoT has been little addressed.

Currently, researchers have proposed intelligent algorithm-based edge computing offloading strategies.

Compared with previous work, this paper has two differences, as follows:

1. The problem of edge computing offloading in an IoT environment is addressed in this paper, and the overall architecture is designed using software-defined thinking. The architecture supports deep reinforcement learning algorithms for training optimal offloading decisions. Currently, we are not aware of any published work that uses the same idea to address the offloading problem of edge computing.
2. Our proposed algorithm obtains environmental information via SDN and creates appropriate rewards that can feed back information about energy consumption, latency, and offload failure of the computed offload policy. The agent is trained multiple times to update the deep neural network in order to derive the optimal offloading policy.

In this paper, we propose a deep reinforcement learning-based offloading method for edge computing in software-defined IoT, ECO-SDIoT, aiming to obtain a task offloading scheme with low latency and energy consumption. We evaluated the performance of the task offloading scheme proposed in this paper and compared it with other offloading schemes. Simulation results show that this scheme can effectively reduce the task completion time and energy consumption. The main contributions of this paper are as follows:

1. To obtain global information, enabling the selection of the best edge server from multiple edge servers to offload tasks a software-defined edge computing framework is proposed. By fusing the control layer with the edge layer, global information about edge servers, network status, and tasks can be obtained. For some IoT devices that cannot support the OpenFlow protocol, this paper introduces the S-MANAGE protocol[15] as a southbound interface protocol between the controller and the data layer to collect network state. The actions output by the agent are parameterized and control messages are sent to the edge server via the interactive interface. The synchronization of state information between edge servers is addressed via the SDN's east-west protocol.
2. To effectively reduce task completion time and energy consumption, a deep reinforcement learning-based compute offloading algorithm, ECO-SDIoT,

is proposed for SD-IoT. Deep reinforcement learning is applied to the edge computation offloading problem in SD-IoT. We take advantage of the global view of SDN to design state space that can fully reflect the dynamic changes of network resources In the ECO-SDIoT, the task model is established to describe the complex task requirements, and the reward is designed to reflect the energy consumption and latency of task offloading, task overload, and task timeout. The prioritised experience replay approach is designed to continuously adjust parameters to get the optimal computational offload solution.

3. To evaluate the performance of our scheme, we chose fault diagnosis in the Industrial Internet of Things (IIoT) as the application background and conducted extensive experiments. Simulation results show that compared with three offloading schemes, namely: distributed computational offloading (DTOS)[16] which does not utilize the SDN architecture, delay-aware computational offloading (LATA)[17] which utilizes the SDN architecture, and task offloading (RJCC)[9] which employs both SDN and deep reinforcement learning, our proposed algorithm can reduce the task completion time as well as the energy consumption.

The remainder of the paper is organized as follows. Section 2 presents a briefly review of related work. Section 3 presents the software-defined architecture, SDEC. Section 4 introduces the deep reinforcement learning-based task offloading algorithm, ECO-SDIoT. Section 5 shows the performance evaluation and simulation results. Finally, Section 6 concludes the paper.

2 Related Work

2.1 Distributed Controller Architecture in SDN

At present, some relatively large networks are usually divided into several smaller subnets when SDN architecture is constructed. Each subnet has an SDN controller, and the controller can only store the local network view[18].

The current view exchange approaches for SDN can be divided into two main categories, namely hierarchical and horizontal architectures[19–22]. The hierarchical architecture is characterized by the fact that there are no east-west interfaces between regional controllers, and the data interaction between them is done through an upper-level controller[19]. The horizontal architecture has east-west interfaces between each of the controllers to complete the necessary data interactions[20–22].

With the distributed controller architecture, the controllers are able to obtain global view information in multiple SDN domains. In this paper, we use the global view information in the controller to perform reinforcement learning and find the optimal edge computing offloading strategy.

2.2 Edge Computing Offloading

Computing offloading is one of the main researches in edge computing, which makes up for the deficiency in computing power and storage resources of end devices and improves task processing in edge computing[23, 24]. Due to the limited computing capacity of the edge server, when the edge server enters a high load state in the face of a large number of offloading requests, the task needs to be offloaded to other edge servers or federate the cloud center for processing[25].

Offloading tasks to other edge servers means that tasks can be offloaded to other edge servers with abundant computing resources for execution when the computing resources of current edge servers are insufficient, to meet the computing resource requirements of tasks[26–28]. Energy consumption and transmission delay are the main considerations in the current computing offloading scheme. Energy consumption is considered as the main factor of edge task offloading in the background of wireless sensor networks[29, 30]. Such methods aim to minimize the total energy consumption while meeting the delay tolerance. Transmission delay is the primary consideration when dealing with computationally-intensive and delay-sensitive tasks[5, 31], and most of the existing distributed joint task offloading and resource allocation schemes ensure delay limitation first.

Furthermore, the global optimization enabled by SDN provides a lot of space for optimizing edge computing offloading. The current research work mainly exploits the characteristics of SDN global scheduling for dynamic task scheduling[32] and combines other algorithms to solve the problems of resource allocation and energy sensing in edge computing.

Some researchers have studied the problem of edge computing offloading based on SDN, and the main application backgrounds are task offloading and resource allocation in vehicular networks[33], computing resource sharing for IoT devices in the context of blockchain[34], and forest fire scenarios[21]. Most of the above methods are based on elaborate heuristics but ignore the lack of adaptive and intelligent handling in the face of dynamic changes in computational demand, network state and resource distribution. To solve this challenging problem, this paper adopts SDN architecture to enhance the control and management of the edge environment, which is greatly enhanced in terms of flexibility and intelligence by balancing many factors such as task execution energy consumption, response time limit, and edge server load balancing degree through deep reinforcement learning.

2.3 Computing Offloading Strategies Based on Deep Reinforcement Learning

Reinforcement learning (RL) introduces ambient intelligence to IoT systems by providing a class of approaches to solve closed-loop problems. And edge computing, a key technology for processing and analyzing massive amounts of IoT data, requires decisions for offloading computational tasks to edge servers. Currently, Deep Reinforcement Learning (DRL) has been used to solve a number of difficult sequential decision problems by combining Reinforcement Learning (RL) with Deep Learning (DL)[9, 35].

The decision optimization for computing offloading in existing work is task-centric and considers the selection of edge servers during task offloading. In this optimization model, the DRL algorithm can be used to make adaptive offloading decisions and optimize the wireless channels[36]. Joint task offloading and bandwidth allocation[37] can also be performed based on DQN to evaluate the total offloading cost and make offloading decisions considering energy consumption, computing capacity and delay.

The Q value of the current state in the Deep Q-Network (DQN) algorithm is estimated as the maximum of the Q values of the next state, which leads to overoptimism due to estimation errors. Van Hasselt et al.[14] proposed an improved Double DQN algorithm, which changes the calculation method of the target value and alleviates the overestimation problem in the DQN algorithm. Therefore, this paper addresses the computing offloading problem in edge environments based on the Double DQN algorithm.

3 Software-Defined Edge Computing Architecture (SDEC)

We extend the idea of software definition to the IoT with edge computing, and the proposed SDEC architecture realizes the global information transfer in the edge environment through the east-west interface of SDN. SDEC mainly makes use of the advantage of SDN to master the state of the whole network, so that the processing of tasks between different edge servers is more efficient, and the specific architecture is shown in Fig.1. The computing, storage and bandwidth resources of these edge servers are abstracted and virtualized through network mapping techniques. Ultimately, edge computing tasks are offloaded by the SDEC controller.

SDEC architecture fully utilizes the east-west interface of SDN, making the connection between controllers into a fast information exchange network. SDEC architecture integrates the controllers and edge server in the same layer, so the controllers can get the distribution of edge computing resources by sharing the global view.

In the SDEC framework, the controller obtains the topology information of the local region through LLDP protocol. In addition, by using SDN-enabled edge devices, information such as remaining computing resources and CPU frequency is updated as uplink messages and uploaded to the controller in the form of flow tables through the Openflow switch. After the PACKET_IN message enters, the controller performs pipeline processing and checks the matching information, packet modification and forwarding rules contained in the flow table. After the above process, a global view of the local edge environment is formed in the controller. Each controller exchanges the local view

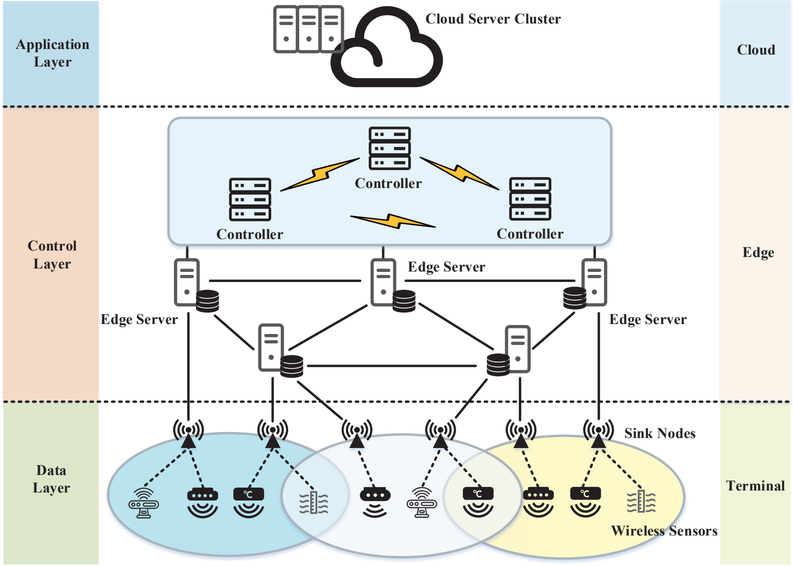

Fig. 1 SDEC architecture. The SDEC architecture has three layers, converging the application, control and data layers of the software-defined architecture with the cloud, edge and end of the edge computing architecture.

information through the east-west protocol to construct the global network view. The global view of the network under SDEC consists of the following main aspects:

1. Network status information, including the status information of controllers, Openflow switches, and wireless sensor devices, as well as the status, bandwidth, and throughput of links.
2. Reachability of edge servers, each controller is connected to the edge servers in its own area. This part mainly includes the status information and reachability of the edge servers.
3. Quality of Service (QoS) of the network, which includes the transmission delay of the network, packet loss rate of the links, delay variation, and cost.

The edge network global view messages contain the information of controller, link, port, edge server, etc. Using a uniform format and encapsulating it into an XML file for east-west delivery makes the format of the network global view message flexible and easily extensible. In this paper, the S-MANAGE protocol[15], East-West Bridge module[38] and Double DQN module are added to achieve the acquisition of the global view of SDN, the interaction of east-west and the perception of edge environment information, as shown in Fig.2.

The modules included in the SDEC controller proposed in this paper can be divided into four categories as follows:

1. Traditional controller-owned modules, such as Network Virtualization Module, Rest API, and so on.

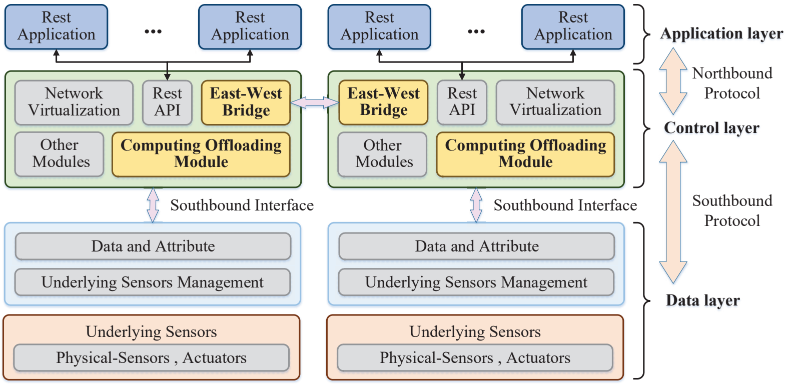

Fig. 2 Computing task offloading based on SDEC. Three main modules are included: 1) East-West Interface Extension Module; 2) Southbound Interface Module; 3) Computing Offloading Module based on Deep Reinforcement Learning.

2. East-West interface expansion module, the East-West Bridge module[38] in Fig.2, is used to enable the East-West interface in all types of controllers.
3. Southbound interface module, it is used to manage and configure underlying network. If IoT devices cannot support the OpenFlow protocol, this paper implements the information interaction by introducing the S-MANAGE protocol[15] as the southbound interface protocol between the SD-IoT controller and the data layer. If IoT devices support the OpenFlow protocol, the SDEC controller receives sensor-centric services via OpenFlow protocol to obtain current edge environment information as state data set for deep reinforcement learning, otherwise it is received via S-MANAGE protocol.
4. Computing offloading module based on deep reinforcement learning. Each SDN controller can be regarded as an agent, which takes the data obtained by the southbound interface as the input of the environment state and obtains the best action, namely the best task offloading strategy, through the deep reinforcement learning algorithm. The specific design of deep reinforcement learning is introduced in Section 4.4. The deep reinforcement learning module is shown in Fig.3. The controller, acting as an agent, obtains the current state of the environment from the edge environment, performs actions and observes rewards, and deposits $\langle state, action, reward \rangle$ samples into the experience replay buffer. Sampling is then performed according to priority for training.

At the beginning of training, in order to avoid overestimation of Q value, we adopted the Double DQN algorithm[14] and designed the deep reinforcement learning module with two deep learning models, called Q-network and target network, where the Q-network is used to predict the optimal values corresponding to each action of the agent in the current environment, while the target network is used to evaluate the optimal values predicted by the Q-network. There is a certain difference between the results obtained by the

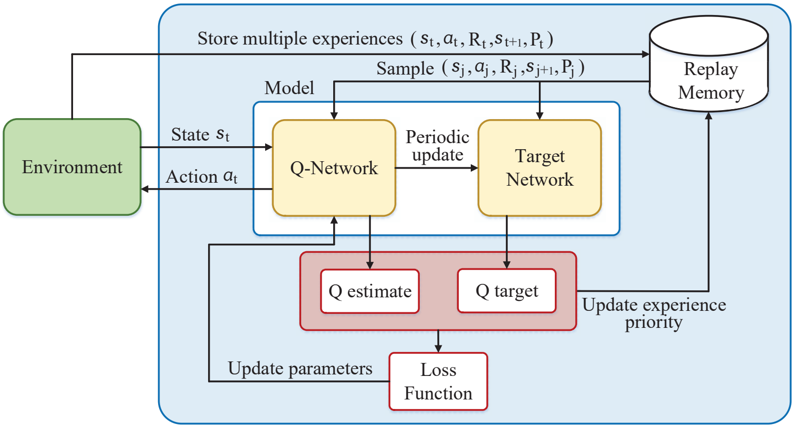

Fig. 3 Double DQN module. This module has two deep neural networks: 1) Q-network; 2) target network.

Q-network and the target network, and the loss function obtained by calculating this difference. The value of loss function is fed back to the Q-network to update the parameters. In addition, the Q-network and the target network are two similar networks, and a delayed parameter update scheme is used, i.e., the model parameters of the target network are updated periodically by the Q-network.

After controllers discover each other, each controller acquires the address of the other controller. Controllers exchange and share global view information periodically or event-driven.

The SDN controller can be regarded as an agent, and outside the agent is regarded as the environment, and the environment changes can be obtained from the SDN global view. Under the SDEC architecture, the controller is able to know the resource distribution in the whole IoT environment in time and find out the optimal edge computing offloading strategy.

4 Deep Reinforcement Learning-based Computing Offloading Algorithm

Before a task is offloaded, it needs to be evaluated. If the resource required by the task exceeds the capacity of the whole edge server, the task is transferred to the cloud platform. Due to limited space, the evaluation process is not described in this paper.

4.1 Delay Model

In this paper, we use T_{all} to represent the completion time of a task. If the task is not executed locally, T_{all} needs to contain the time the task is transmitted to the edge server (T_{send}), the time the task is executed on the edge server

(T_{exe}), the queuing delay (T_{que}), and the time the task execution result is transmitted back to the local device (T_{ret}).

Then the completion time of the task T_{all} satisfies:

$$T_{all} = T_{send} + T_{que} + T_{exe} + T_{ret} \tag{1}$$

where T_{send}, T_{que}, T_{ret} can be obtained by counting the network data respectively. T_{exe} indicates the execution time of the task on the edge server, and its value is determined by the CPU processing frequency of the edge server. If there are heterogeneous edge servers in the network, T_{exe} is defined as follows:

$$T_{exe} = W_{task}/f_{edge} \tag{2}$$

where W_{task} denotes the computational workload of the edge task and f_{edge} denotes the working frequency of the processing units of the edge server.

4.2 Energy Consumption Model

In the SDEC architecture, the underlying sensor network only undertakes data collection and data forwarding, and the computation and processing tasks are performed by the edge server. Therefore, the overall energy consumption is divided into two parts: wireless transmission energy consumption and task processing energy consumption.

4.2.1 Wireless Transmission Energy Consumption

Most of the IoT end devices are energy constrained, so we need to monitor the sending energy consumption E_{send} consumed by the end to send tasks to the edge server, and the receiving energy consumption E_{rcv} consumed by the end to receive the computing results from the edge server.

According to the [31], the transmitting energy consumption of the wireless sensor is:

$$E_{send} = E_{elec} + \varepsilon_{amp} \times d_s^2 \tag{3}$$

where E_{elec} denotes the energy consumption caused by transmitting and receiving data, ε_{amp} denotes the power amplifier power consumption required to boost the transmitting power of the wireless sensor. The energy consumption of the wireless sensor to receive data is shown in formula (4), where the energy consumption of the sensor to receive data is independent of the distance.

$$E_{rcv} = E_{elec} \tag{4}$$

4.2.2 Task Processing Energy Consumption

When tasks are offloaded to the edge server, the energy consumption of the edge server to process the tasks depends mainly on the CPU power consumption of the edge server and the execution time of the tasks.

The energy consumption for task processing is:

$$E_{exe} = P_{cpu} \times T_{exe} \tag{5}$$

where P_{cpu} denotes the power consumption of the CPU of the edge server, which is positively related to its operating frequency. The power consumption of the CPU can be calculated by:

$$P_{cpu} = \lambda V^2 f_{edge} \tag{6}$$

where λ denotes a constant, which is determined by the process and design of the processing unit. V denotes the voltage during operation, and f_{edge} denotes the operating frequency of the processing unit on the edge server.

The energy consumption E_{all} generated during task offloading is defined as the sum of the wireless transmission energy consumption of the task and the task processing energy consumption. The transmission energy consumption is considered as the transmit energy consumption E_{send} and the receive energy consumption E_{rcv} consumed by the end devices, while the computational energy consumption E_{exe} is mainly generated in the edge servers. and thus E_{all} can be expressed as:

$$E_{all} = E_{send} + E_{exe} + E_{rcv} \tag{7}$$

The transmission energy consumption of the edge servers is not considered in this paper because it is negligible for the other energy consumption generated by the servers.

4.3 Task Model

There are some properties exist within the edge tasks themselves, including the computational resources required by the tasks, the deadline of the tasks, and the priority levels, etc.

The deadline of the task can indicate the urgency of the edge task. We define the task priority Pr to reflect the importance of different edge tasks and the sensitivity to latency. This paper comprehensively describes the current edge tasks from multiple perspectives. We construct the task model with the following components:

(1) C_{need}, the computational resources required by the task,
(2) ddl, the current task completion deadline,
(3) Pr, the priority of the current task.

The task model can be represented as follows:

$$task = \{C_{need}, ddl, Pr\} \tag{8}$$

Selecting task offloading schemes based on the characteristics of the tasks and the distribution of edge resources is the main research objective of this paper. To address the problem of computational resource differentiation in edge environments, the above model is constructed to describe the network performance and task requirements using a multidimensional perspective and to make fast decisions using deep reinforcement learning to minimize the execution cost of edge tasks.

4.4 Computing Offloading Problems

The nearest edge server to the task may not have abundant remaining computational resources, so the information about the computational capacity, remaining computational resources, and transmission distance of the candidate edge servers in the environment needs to be input into a deep neural network for learning to obtain the optimal task offloading action. This paper proposes a scheme (ECO-SDIoT) to solve the problem of offloading optimisation of edge tasks under multiple controllers. By observing the computational resources and network states in the edge environment, task offloading actions are selected based on deep reinforcement learning to maximize their cumulative rewards. When an edge task to be offloaded is generated, the algorithm needs to select the target edge server so that the edge devices can offload the task to the corresponding edge server for execution. We assume that the number of uninstalled tasks is N and the total number of edge servers is M. The selection of actions in different environments is achieved by deep reinforcement learning, and this process has the following key elements.

4.4.1 Objective Function *Des*

The objective function Des consists of the following two components, which are the total completion time T_{all} for the task to be offloaded to the edge server j and the total energy consumption E_{all} during the task offloading process, calculated as follows:

$$T_{all} = \sum_{j=1}^{M} \left(T_{send}^{j} + T_{que}^{j} + T_{exe}^{j} + T_{rcv}^{j} \right) \tag{9}$$

$$E_{all} = \sum_{j=1}^{M} \left(E_{send}^{j} + E_{exe}^{j} + E_{rcv}^{j} \right) \tag{10}$$

The optimization objective of the computational offloading strategy is to minimize task completion time and energy consumption as the network state and task demand dynamically change, and the objective function is:

$$Des = \arg\min_{S,A} \left(T_{all} + E_{all} \right) \tag{11}$$

In this paper, the convergence of total energy consumption and total delay or episode=20,000 is set as the end condition of deep learning iteration state.

4.4.2 State Space *S*

The state is the information about the algorithm's perception of the environment, and the state space S can be composed of the following parts:

(1) C_{rem}^{j}, remaining computing resources of edge server j,
(2) f_{edge}^{j}, operating frequency of the processing unit of edge server j,
(3) $d_{s}^{i,j}$, transmission distance from task i to edge server j,
(4) $bw^{i,j}$, link bandwidth from task i to edge server j obtained from the global view,
(5) plr, packet loss rate in the current network topology,

The state space S can be expressed as:

$$S = \left\{ C_{rem}^{j}, f_{edge}^{j}, d_{s}^{i,j}, bw^{i,j}, plr \right\} \tag{12}$$

The remaining computing resources of the jth edge server can be defined as follows:

$$C_{rem}^{j} = 1 - L_{ave}^{j} / \left(N_{cpu}^{j} \times PN_{cpu}^{\max} \right) \tag{13}$$

where L_{ave}^{j} denotes the average number of processes of the jth edge server system, N_{cpu}^{j} denotes the number of CPUs of the jth edge server, and PN_{cpu}^{max} denotes the maximum number of processes per CPU. The smaller the value of C_{rem}^{j}, the fewer computing resources remain on the edge server, and when the value of C_{rem}^{j} is less than 0, the system enters overload.

4.4.3 Action Space *A*

The action space A contains a series of actions, consisting of a set of binary values, denoted by $a^{i,j}$ whether end task i is offloaded to edge server j. It can be defined as follows:

$$A = \left\{ a^{1,1}, a^{1,2}, \cdots, a^{i,j}, \cdots, a^{N,M} \right\} \tag{14}$$

where $a^{i,j} \in \{0, 1\}$, $a^{i,j} = 0$ means that end task i isn't offloaded to edge server j. Conversely, $a^{i,j} = 1$ means that end task i is offloaded to edge server j. The actions are initially generated randomly by the algorithm and finally the optimal action is derived by deep reinforcement learning.

4.4.4 Reward *R*

Agents in reinforcement learning rely on rewards to evaluate the effectiveness of actions and to further improve strategies, and different actions usually imply that different rewards will be obtained. According to the target function of the computing offloading problems, the reward R of deep reinforcement learning considers the total energy consumption and the completion time of computing offloading, and is defined as:

$$R = \begin{cases} 0 & \text{, if task lose} \\ \frac{1}{T_{all}} + \frac{1}{E_{all}} & \text{, if task accomplished} \\ -1 & \text{, if task timeout} \end{cases} \tag{15}$$

From formula (15), it can be seen that the reward R decreases as the total completion time and total energy consumption of the current task increases.

When a task timeout occurs, the reward is set to -1 to generate negative feedback. If a task is lost due to the saturation of the edge server computational queue, the reward R is set to 0.

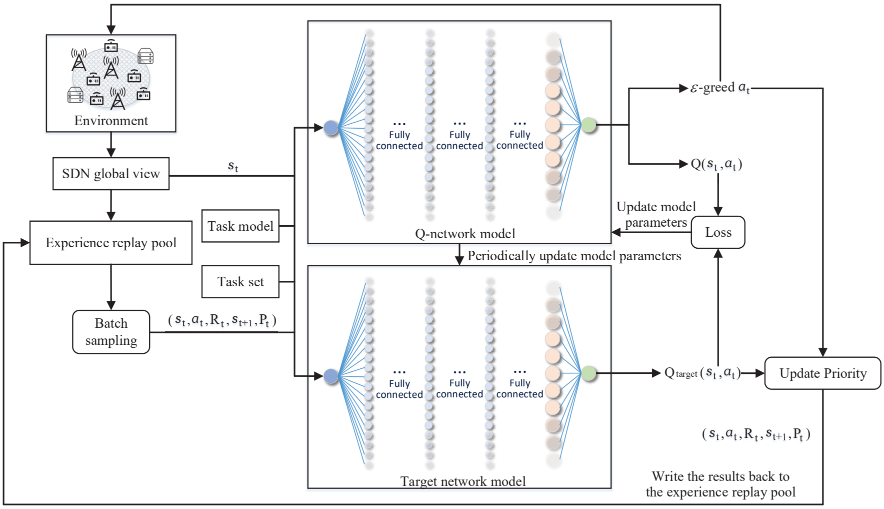

Fig. 4 The improved Double DQN model. Both the Q-network and the target network are three-layer neural network structures. The improved Double DQN model is based on the SDN structure to obtain multidimensional state information and rewards, which are stored in an experience buffer. Priority-based sampling information and task information are used as input to derive optimal actions based on the ECO-SDIoT algorithm.

The improved Double DQN model contains three layers of architecture as shown in Fig.4. The input layer feeds the current environment features obtained from the global view as well as the task model into the neural network. Different from the traditional Double DQN, the state information involved in this paper does not contain two-dimensional pictures, so instead of using a convolutional layer, the hidden layer is designed as a fully connected structure where each node is connected to all nodes in the previous layer to synthesize the feature information extracted from the previous layer. The purpose of designing the fully connected structure in the hidden layer is to map the features of distributed network into the sample labeling space, approximating the action value function $Q(s, a)$. The value of $Q(s, a)$ is the expected future reward for taking action a in state s. The agent's purpose is to interact with the environment by choosing actions that maximize future rewards. Therefore, the optimal action value function $Q^*(s, a)$ is defined as the maximum expected reword obtained by following any strategy π under some sequence s and then performing some action a[43].

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[R_t \mid s_t = s, a_t = a, \pi\right] \tag{16}$$

In order to solve the overestimation problem, Double DQN uses two sets of Q-networks for network parameter updating. Q-network is responsible for selecting actions, with parameters θ. Target network is responsible for calculating updated target values with parameters θ'. The output layer obtains the $Q(s,a)$ values for each action of the current environment based on the features mapped in the hidden layer. To estimate the action value function, it is common to use a function approximator, $Q_{target}(s,a) \approx Q^*(s,a)$. The action with the maximum value in the Q-network is used to calculate the action value function of the target network $Q_{target}(s,a)$. At time step t, $Q_{target}(s,a)$ is defined as follow, where γ represents the discount rate.

$$a^* = \underset{a}{\operatorname{argmax}} Q\left(s_{t+1}, a, \theta\right) \tag{17}$$

$$Q_{target}\left(s_t, a_t, \theta'\right) = R_t + \gamma \times Q_{target}\left(s_{t+1}, a^*, \theta'\right) \tag{18}$$

The update of deep neural network parameters in the algorithm requires defining the loss function and then updating the parameters by gradient descent. Comparing the estimated value $Q(s,a)$ obtained from the deep neural network and the target value $Q_{target}(s,a)$ calculated from the target network. The loss function is calculated and the model parameters in the Q-network and the target network are updated. The loss function is calculated as follows:

$$L(\theta) = \mathbb{E}\left[\left(Q_{target}\left(s, a, \theta'\right) - Q(s, a, \theta)\right)^2\right] \tag{19}$$

The specific deep reinforcement learning-based computing offloading process in IoT is shown in the ECO-SDIoT algorithm. The algorithm can obtain the computational resources and the performance of the edge server, etc. from the global view of the controller. To begin with, N tasks ordered by priority, network computing resources, and network state are all imputed to ECO-SDIoT. Then we define a three-layer fully connected layer as the hidden layer of the Q-network and the target network, replicate the Q-network model to the target network after creating it, and initialize the network with randomly generated weights θ.

In addition, we select a random action with probability ε, or select an action that maximizes the value of $Q(s,a)$. The action a_t is executed in the simulation under the network state s_t and the reward R_t is calculated according to formula (15), then the network state is updated as s_{t+1}. Furthermore, we store s_t, a_t, R_t, s_{t+1} into the experience replay buffer. The initial value of the priority of each sample is set to P. If the experience replay buffer is full, the earliest data items will be deleted.

When the offloading policy of N tasks is trained, a batch sample is taken from the experience replay buffer according to the priority. For each sample, $Q^u_{target}(s,a)$ represents the target network action function value of task u, and it is calculated based on formula (18). We compute the loss function between $Q^u(s,a)$ and $Q^u_{target}(s,a)$ and update the parameters θ based on the gradient descent. As well as, the prioritized experience replay mechanism[39] is used to sample by priority and to maintain the independent homogeneous distribution of the neural network training set. We recalculate the priority of current sample and update it to the experience play buffer. After the fixed times of update, the model parameter θ is updated to θ'.

Algorithm 1 ECO-SDIoT

```
Input: Given $N$ tasks that have been sorted and need to be offloaded on edge servers; Given global view information (including edge server information and network status) in tuple form
Output: The optimal task allocation scheme
Total reward is defined as 0
Random initialization of model parameter $\theta$
Initialize target network parameters $\theta' = \theta$
Initialize random data $\varepsilon$
Define priority parameter $\eta$ and integer $\xi$
Initialize experience priority as $P$
for episode $= 1, E$ do
    for $h = 1, H$ do
        With probability $\varepsilon$ select a random action $a_h$
        Otherwise select $a_h = \underset{a}{\operatorname{argmax}} Q\left(s, a, \theta\right)$
        Execute task offloading action $a_h$ in emulator
        Calculate the reward $R_h$ according to the input task model and the definition of the reward equation (15)
        Collect the state $s_{h+1}$ through global view after the action is performed
        Store transition $(s_h, a_h, R_h, s_{h+1}, P_h)$ in experience replay buffer, where $P_h = P$
    end for
    for $t = 1, T;\ u = 1, N;\ t = u$ do
        Extract $b$ experiences based on priority from the experience replay buffer
        Calculate $Q^u_{target}\left(s_t, a_t, \theta'\right) = \begin{cases} R_t \quad, \text{ if } s_{t+1} \text{ is terminal} \\ R_t + \gamma \times Q^u_{target}\left(s_{t+1}, a_t, \theta'\right), \text{ otherwise} \end{cases}$
        Calculate the absolute error $\omega_t = |Q^u_{target}(s_t, a_t, \theta') - Q^u(s_t, a_t, \theta)|$
        Calculate the loss function
            $$L(\theta) = \mathbb{E}\left[\left(Q^u_{target}\left(s_t, a_t, \theta'\right) - Q^u\left(s_t, a_t, \theta\right)\right)^2\right]$$
        Update model parameter $\theta = \theta - \alpha \nabla_\theta L(\theta)$
        Calculate the priority of this round $P_t = \frac{(|\omega_t| + \xi)^\eta}{\sum_{k=1}^{b} (|\omega_k| + \xi)^\eta}$
        Store experience $(s_t, a_t, R_t, s_{t+1}, P_t)$ to the experience replay buffer
        return Optimal task allocation policy
    end for
    Periodically update target Q-network parameters $\theta' = \theta$
end for
```

After completing the offloading decision, the controller passes the parameterized command to the terminal device, which sends the task to the edge server specified in the decision for execution. At the same time, the global view needs to be updated via an east-west protocol between controllers to ensure its consistency.

5 Simulation Results and Analysis

In this section, we conduct simulation experiments on computing offloading under software definition. The experimental process is divided into three parts:

(i) Investigate the variation of the loss function during the iteration of this deep reinforcement learning algorithm.

(ii) The proposed ECO-SDIoT is compared with three algorithms, which are distributed computing offloading (DTOS)[16], delay-aware computing offloading (LATA)[17] and reinforcement learn-based computing offloading (RJCC)[9].

(iii) Investigate the impact of task data size on the computing offloading performance.

5.1 Experimental Setup

We use the EdgeCloudSim[40] to simulate the edge computing environment. EdgeCloudSim is a simulation environment provided for edge computing scenarios, in which experiments can be conducted considering both computational and network resources. This experiment simulates different numbers of edge servers running at the same time, and the performance of edge devices and the congestion of the link will be randomly generated. The experiments simulate the global view passing of SDN east-west architecture with NS3, and the global view information generated by the simulation is encapsulated into a configuration file, and the configuration is read by the network module of EdgeCloudSim. To implement deep Q-network, this paper uses the TensorFlow and Keras libraries under Python, running under a multi-core CPU server equipped with a 48-core Intel XeonGold 5118 processor.

In ECO-SDIoT algorithm, the current state of edge environment in the document encapsulated by global view, task model and samples sampled from experience playback pool are taken as input, alternative actions are taken as output layer, the optimal actions are saved as parameters, and EdgeCloudSim executes the optimal task offloading scheme. For the hidden layer, various choices can be made depending on the information of the input layer. In this paper, a three-layer fully connected internal product layer is used as the hidden layer of the deep neural network. The ReLU function is used as the activation function in the deep neural network, and the deep neural network is applied to the simulation environment of edge computing to verify the efficiency of task processing. The settings of the simulation parameters are shown in Table 1.

The training process of deep neural networks requires a large set of tasks, and we use fault diagnosis in the Industrial Internet of Things (IIoT) as

Table 1 Key Simulation Parameters

Parameter	Value
Number of Edge Server	20
Transmission bandwidth	[100-300] Mbps
Average task data size	50MB
Required CPU cycles of computation task	[0.5-4] Gcycle/s
The maximum delay constraint	[0.2-1] s
Greed	0.90
Learning efficiency	0.80
Attenuation degree	0.90
Hidden size	128
Replay memory buffer size	200GB
Batch size	64MB

an application scenario for this paper. The MFPT bearing fault diagnosis dataset[41] is invoked to randomly generate 10,000 independent tasks with an arrival rate of 0-10 per second. According to the randomly generated edge server computing capacity of EdgeCloudSim, the corresponding computing resources are allocated in the server, and these independent tasks are offloaded to the virtual edge server according to the method in this paper, and deep neural networks based on Gaussian Bernoulli restricted Boltzmann machines (GBRBMs-DNN)[42] are deployed in the virtual edge server to process and analyze the bearing failure data and make decisions based on the analysis results. The performance indicators in Section 5.2 are the simulation results when the fault diagnosis accuracy reaches 98%.

With the increasing number of training rounds, it can be found that the loss function between the Q-network and the target Q-network decreases during the selection of the optimal computing offloading strategy, and the generated reward stabilizes at a higher value as the number of training rounds increases to a certain level. As shown in Fig.5, this paper compares the Q-network reward enhancement for three different learning rate α, and it can be found that as the learning rate increases, the reward can be maintained at a higher level more quickly. When the learning rate of the deep reinforcement learning algorithm is 0.5, it takes about 1000 rounds of training to maintain the reward at a high level. When the learning rate is 0.9, only 400 rounds of training are needed to quickly make the reward converge to a higher value.

State, action, next state, reward, and experience priorities are updated to the experience replay buffer after each training. According to the priority of batch sampling, the agent can learn more important experience, and improve the sampling efficiency. In addition, we improve the sampling efficiency by running multiple controllers as multiple agents simultaneously through a distributed parallel sampling approach. After continuous optimization feedback, the accuracy of the controller to obtain the optimal action in the current environment is improved, and therefore the reward function is also improved.

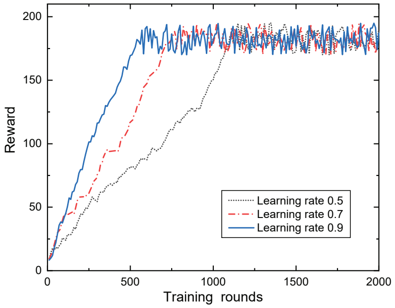

Fig. 5 Reward variation for different learning rates

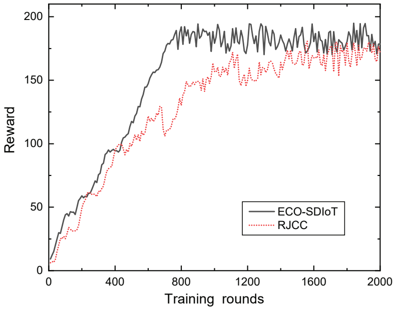

Fig. 6 Reward comparison between the ECO-SDIoT and RJCC

5.2 Performance Evaluation

In this section, we compare the ECO-SDIoT algorithm with the distributed task offloading strategy (DTOS)[16], the delay-aware task offloading strategy (LATA)[17], and the reinforcement learning-based joint communication and computational resource offloading mechanism (RJCC)[9]. We evaluate the performance of the ECO-SDIoT algorithm by comparing four performance indicators of computing offloading: unit task processing latency (UTPD), edge server load balancing degree (ELBD), task processing energy consumption (TPEC), and task completion rate (TCR), respectively.

5.2.1 Comparison Objects

Distributed task offloading strategy (DTOS)[16] is a method to offload distributed tasks to a cluster of low-load base stations in a mobile edge computing environment. DTOS solves the distributed task offloading problem in edge environments by modeling the communication and computational resources of local low-load base station clusters using a game-theoretic approach.

Latency-aware task offloading policy (LATA)[17] is a method to make task offloading decisions through SDN to reduce task processing delays. LATA is characterized by its full consideration of network load and load balancing to improve the efficiency of edge task offloading.

The reinforcement learning-based joint communication and computational resource offloading mechanism (RJCC)[9] is divided into two parts when performing task offloading, the Q-learning-based online offloading algorithm and the Lagrange-based migration algorithm. It transforms the long-term optimization problem to two sub-problems through Lyapunov optimization theory to jointly optimize the computing tasks in the edge environment offloading.

Reinforcement learning is used in both the RJCC and the ECO-SDIoT algorithm proposed in this paper. At the same learning rate (0.9), we compare the reward of the ECO-SDIoT and RJCC during training and find that the reward of ECO-SDIoT converges faster and converges to a higher value. Fig.6 demonstrates that the reward of ECO-SDIoT algorithm converges after only 700 training rounds, whereas the RJCC converges after 1000 training rounds.

It is due to the fact that the RJCC uses Q-learning to find the optimal Q value, which decreases the querying speed when the state space grows larger. In addition, the RJCC does not consider the overestimation problem, and the final convergence value of reward is smaller than the ECO-SDIoT. ECO-SDIoT is based on the idea of Double DQN, which uses deep neural networks to compute $Q(s,a)$, uses two deep neural networks to solve the overestimation problem, and uses prioritized experience replay for sampling, so that the reward reaches convergence faster and has a higher convergence value than RJCC.

5.2.2 Unit Task Processing Delay (UTPD)

UTPD is the time interval between the start of computing offloading and the complete the task (ms). The UTPD contains two components, the task offloading time and the task computation time. In the ECO-SDIoT algorithm, the task offload time in UTPD includes the decision time of the task offload and the task transfer time, while the task computation time is spent by the edge server to execute the task. We compare the UTPD of four methods, DTOS, LATA, RJCC, and ECO-SDIoT, from the perspectives of both the number of tasks and the amount of computation per task, and the experimental results are shown in Fig.7 and Fig.8.

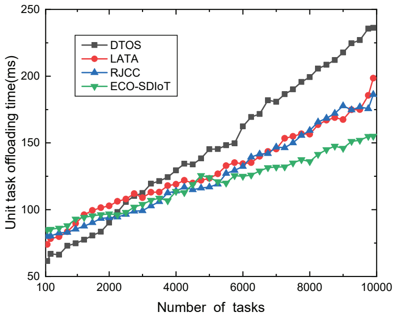

Fig. 7 The relationship between the number of tasks and the unit task offloading time

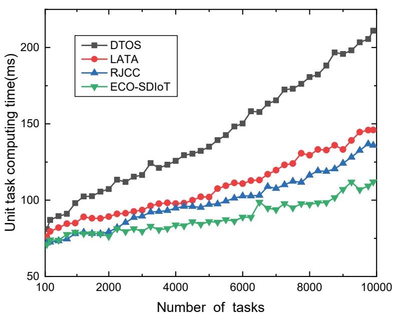

Fig. 8 The relationship between the number of tasks and the unit task computing time

As illustrated in Fig.7 that the delay of the four methods in the offloading and computation phase increases with the increase of the number of offloading tasks. Among them, ECO-SDIoT has the most gradual growth, which is more noticeable in terms of computation delay. DTOS distributes the task to local low load base station clusters, making the task offloading time the shortest when the number of tasks is less. Meanwhile, as shown in Fig.8 when the number of tasks increases, the task computation time of DTOS increases rapidly due to task load unbalance. Although the LATA also offloads tasks through the controller based on the SDN global view, it only offloads tasks to the edge servers with lower load. The ECO-SDIoT algorithm takes into account the computational power of the edge server in the state and the time consumption during task transfer in the reward, so its UTPD is lower than LATA. When the number of tasks is small, the decision time of RJCC is less than that of ECO-SDIoT. However, as the number of tasks increases, the expansion of the Q-table leads to a rapid increase in query time for the Q-learning based RJCC algorithm.

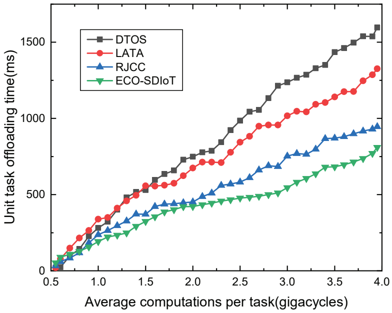

Fig. 9 The relationship between unit task calculation volume and the unit task offloading time

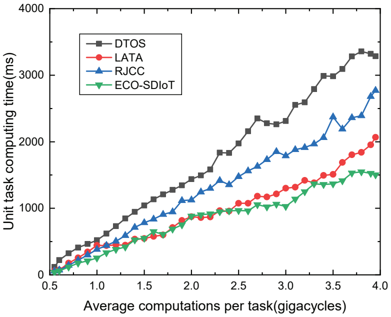

Fig. 10 The relationship between unit task calculation volume and the unit task computing time

Then we vary the amount of computation per task to investigate the impact of task complexity on the computational offload policy. As shown in Fig.9 and Fig.10, the computing time ECO-SDIoT has a more pronounced advantage over the other three algorithms as the computational volume of the task increases, indicating that ECO-SDIoT is able to make better decisions as the task's demand for computational resources increases. Despite RJCC further optimises the task offload solution after offloading tasks using a Lagrangian-based migration algorithm, the cost of task migration is also reflected in the task offload time.

5.2.3 Load Balancing Degree for Edge Servers (ELBD)

ELBD is the degree of task load balancing among the edge servers in one or more regions, and this parameter reflects whether the computational resource of multiple edge servers is distributed in a balanced manner. ELBD is calculated as shown in formula (20), where SL_i represents the load rate of the ith edge server and the $\overline{SL}$ represents the average load rate of the edge servers in the edge environment.

$$ELBD = \frac{\sum_{i=1}^{M} \left(SL_i - \overline{SL}\right)^2}{M-1} \tag{20}$$

We compared the ELBD of four methods, DTOS, LATA, RJCC, and ECO-SDIoT, by varying the number of tasks and the average task computation, and the experimental results are shown in Fig.11 and Fig.12.

As the number of tasks gradually increases from 100 to 10,000, the Load balancing degree for edge servers (ELBD) of ECO-SDIoT, RJCC, and LATA

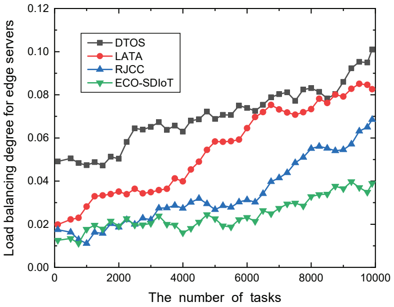

Fig. 11 The load balancing degree for edge servers in different task numbers

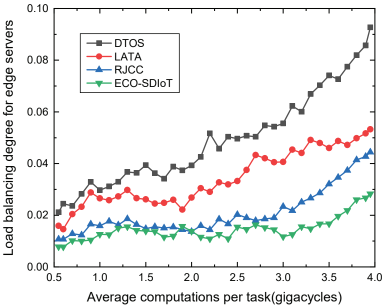

Fig. 12 The load balancing degree for edge servers in different average task computations

all remain lower than 0.08, while DTOS increases rapidly as the number of tasks increases. This is because both ECO-SDIoT- and LATA consider the server workload in the edge environment holistically through the SDN global view, while DTOS only considers the local base station group load situation. As the number of tasks increases, the load of tasks is gradually non-uniform, resulting in a higher ELBD for DTOS. In addition, we find that when the number of tasks is small, the optimization effect of ECO-SDIoT is not obvious compared with LATA and RJCC, and even when the number of tasks is less than 2000, the ELBD of the ECO-SDIoT is higher than that of LATA. The reward model in the ECO-SDIoT algorithm considers the task overload. When the number of tasks is small, the chance of task overload is low, and as the number of task increases, task overload is able to provide feedback through the reward. RJCC designs a Lagrange-based task migration algorithm that balances the workload between edge nodes by migrating tasks from high-load edge nodes to relatively low-load nodes. Therefore, RJCC's ELBD is close to ECO-SDIoT.

5.2.4 Task Processing Energy Consumption (TPEC)

By varying the size of the average task computation and the average amount of data, we evaluate the impact of task complexity on the energy consumption of the computational offload policy.

The processing energy consumption of the task contains the energy consumption generated by the task transmission and execution. In this part of the experiment, we compare the difference in total energy consumption between the four methods, DTOS, LATA, RJCC, and ECO-SDIoT, when completing the edge tasks.

As can be seen in Fig.13 and Fig.14, the ECO-SDIoT outperforms the other algorithms in terms of processing energy consumption of the task. The reason is that the operating frequency of the processing units on the edge server, the transmission distance from the edge device to the edge server, and other factors also be considered in the state space of ECO-SDIoT, so that it can continuously adjust the computing offloading decision according to the

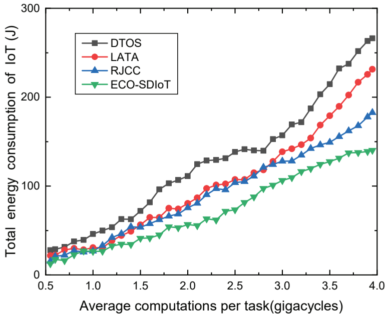

Fig. 13 Task processing energy consumption in different average task computations

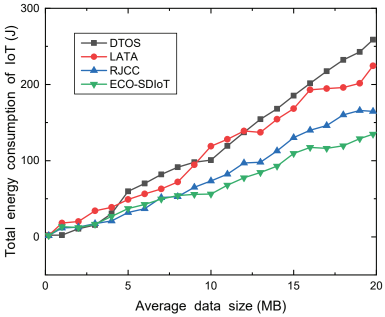

Fig. 14 Task processing energy consumption in different average data sizes

rewards and losses. The energy consumption is also considered in the reward function to reduce the unnecessary energy consumption generated by the task during the offloading process.

5.2.5 Task Completion Rate (TCR)

TCR is the probability that a task does not time out or get lost during offloading and execution. This parameter is used to determine whether the task is offloaded to the appropriate edge server in a timely manner. We compared the TCR of four methods, DTOS, LATA, RJCC, and ECO-SDIoT, and the experimental results are shown in Fig.15 and Fig.16.

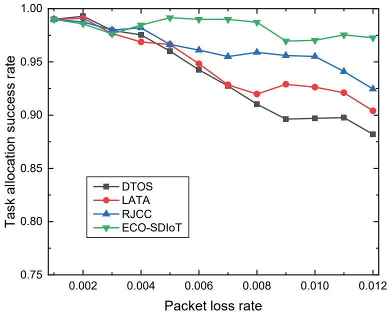

Fig. 15 Task completion rate in different packet loss rates

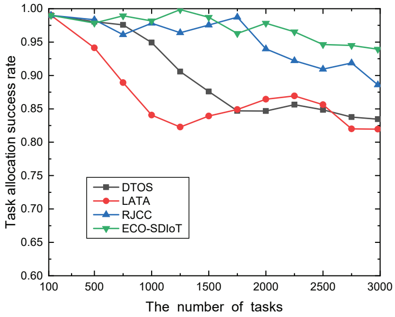

Fig. 16 Task completion rate in different task numbers

By varying the packet loss rate and the number of tasks, we evaluate the task completion rates of four algorithms.The results show that ECO-SDIoT has a significantly better TCR than DTOS and LATA. When the packet loss rate is low, the task completion rates of four methods are at a high level, and as the packet loss rate gradually increases, the task completion rates of DTOS, LATA, and RJCC all have a significant downward trend, and only the TCR of ECO-SDIoT is relatively stable. Because ECO-SDIoT utilizes the global view of SDN, it can obtain the link condition of the edge network more effectively and input it into the deep neural network as the current state of the network, which further improves the TCR.

ECO-SDIoT not only models the remaining computational resources of the edge server into the state space, but also feeds back task timeouts or losses through rewards and uses a prioritised experience replay approach to continuously adjust parameters to get the optimal computational offload solution. Therefore, as shown in Fig.16, the TCR of ECO-SDIoT decreases the slowest among the four methods as the number of tasks increases.

6 Conclusion

In order to achieve the optimization goals of low energy consumption, low latency, and load balancing for edge computing offload policies in an environment where the network state and task demands are dynamically changing, we propose a software-defined edge computing architecture and present a compute offloading algorithm named ECO-SDIoT to offload computing tasks to the optimal edge server.

First, we propose a software-defined edge computing framework in the IoT environment. The controller in the framework is able to obtain a global view to generate the state of the environment, and the controller has the programmable capability to become an agent to provide support for deep reinforcement learning. Moreover, our proposed algorithm obtains environmental information via SDN and creates appropriate rewards that can feed back information about energy consumption, latency, and offload failure of the computed offload policy. The agent is trained multiple times to update the deep neural network in order to derive the optimal offloading policy. Finally, the simulation results show that the proposed algorithm achieves better performance in terms of unit task processing delay, load balance of edge servers, task processing energy consumption, and task completion rate compared with the three related works.

In our future work, we intend to investigate the controller cooperative task allocation strategy in the software-defined IoT edge computing framework.

Declarations

Ethics approval and consent to participate. Not applicable.

Consent for publication. All authors approved the final manuscript and the submission to this journal.

Availability of data and material. The data and material used or analysed during the current study are available from the corresponding author on reasonable request.

Competing Interests. The authors declare that there is no conflict of interest regarding the publication of this paper.

Abbreviations. The relevant abbreviations involved in this paper are shown in Table 2.

Table 2 Abbreviation Table

Abbreviation	Full Form
DL	Deep Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DTOS	Distributed Task Offloading Strategy
ECO-SDIoT	Edge Computing Offloading Algorithm in Software-defined IoT
IoT	Internet of Things
IIoT	Industrial Internet of Things
LATA	Latency-aware Task Offloading Policy
ELBD	Load Balancing Degree for Edge Servers
QoS	Quality of Service
RJCC	Joint Communication and Computational Resource Offloading Mechanism
RL	Reinforcement learning
SDN	Software Define Network
SDEC	Software Defined Edge Computing Architecture
TCR	Task Completion Rate
TPEC	Task Processing Energy Consumption
UTPD	Unit Task Processing Delay

Funding. This work is jointly supported by National Natural Science Foundation of China (Grant No. 62076006) and the Natural Science Research Project of Colleges and Universities in Anhui Province of China (Grant No. KJ2020A0300) and the Huainan Municipal Science and Technology Project(Grant No. 2021A243)

Authors' contributions. The main idea of this paper was proposed by Xiaojuan Zhu and Tianhao Zhang. The algorithm design and the experimental design were jointly completed by Xiaojuan Zhu and Tianhao Zhang. The experimental performance analysis was completed by Tianhao Zhang, Bao Zhao and Cai Wu. The article was co-written by Xiaojuan Zhu and Tianhao Zhang. And the writing guidance, English polishing, and funding project are completed by Xiaojuan Zhu, Jingwei Zhang, and Shunxiang Zhang. All authors read and approved the final manuscript.

Acknowledgments. This work is supported by National Natural Science Foundation of China (Grant No. 62076006) and the Natural Science Research Project of Colleges and Universities in Anhui Province of China (Grant No. KJ2020A0300) and the Huainan Municipal Science and Technology Project(Grant No. 2021A243). The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in our work. We would also like to thank all teachers and students in our laboratory for helpful discussions.

References

[1] Premsankar G., MD Francesco, and Taleb T.: Edge Computing for the Internet of Things: A Case Study. IEEE Internet Things J. 5(2), 1275-1284 (2018)

[2] Geng Y., Yi Y., and Cao G.: Energy-Efficient Computation Offloading for Multicore-Based Mobile Devices. In: International Conference on Testbeds & Research Infrastructures. Springer(2018)

[3] Katayama Y., Tachibana T.: Collaborative Task Assignment Algorithm to Reduce Total Response Time in MEC Platform. In: International Conference on Information and Education Technology (2020)

[4] Wu Y., Shi B., and Qian L P., et al.: Energy-Efficient Multi-task Multi-access Computation Offloading Via NOMA Transmission for IoTs. IEEE Trans. Ind. Inf. 16(7), 4811-4822 (2020)

[5] You Q., Tang B.: Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. J. Cloud Comput-Adv. S. 10(41), 609-620 (2021)

[6] Naouri A., Wu H., and Nouri N. A., et al.: A Novel Framework for Mobile-Edge Computing by Optimizing Task Offloading. IEEE Internet Things J. 8(16), 13065-13076 (2021)

[7] Qu G., Wu H., and Li R.: DMRO: A Deep Meta Reinforcement Learning-Based Task Offloading Framework for Edge-Cloud Computing. IEEE Trans. Netw. Serv. Manage. 18(3), 3448-3459 (2021)

[8] Shahidinejad A., Farahbakhsh F., and Ghobaei-Arani M., et al.: Context-aware multi-user offloading in mobile edge computing: a federated learning-based approach. J. Grid Comput. 19(2), 1-23 (2021)

[9] Xu S., Liu Q., and Gong B., et al.: RJCC: Reinforcement-Learning-Based Joint Communicational-and-Computational Resource Allocation Mechanism for Smart City IoT. IEEE Internet Things J. 7(9), 8059-8076 (2020)

[10] Hu F., Hao Q., and Bao K.: A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. IEEE Commun Surv Tut. 16(4), 2181-2206 (2014)

[11] Hu P., Chen W., and He C., et al.: Software-Defined Edge Computing (SDEC): Principle, Open IoT System Architecture, Applications, and Challenges. IEEE Internet Things J. 7(7), 5934-5945 (2020)

[12] Guo Y., Wang W., and Zhang H.: Traffic Engineering in Hybrid Software Defined Network via Reinforcement Learning. J. Netw. Comput. Appl. 189(103116) (2021)

[13] Younus M. U., Khan M. K., and Bhatti A. R.: Improving the Software-Defined Wireless Sensor Networks Routing Performance Using Reinforcement Learning. IEEE Internet Things J. 9(5), 3495-3508 (2021)

[14] Hasselt H. V., Guez A., and Silver D.: Deep Reinforcement Learning with Double Q-Learning. In: Proceedings of the AAAI Conference on Artificial Intelligence (2016)

[15] Nguyen T., and Hoang D. B.: S-MANAGE Protocol For Software-Defined IoT. In: 28th International Telecommunication Networks and Applications Conference (2018)

[16] Li Y., and Jiang C.: Distributed task offloading strategy to low load base stations in mobile edge computing environment. Comput. Commun. 164, 240-248 (2020)

[17] Sscg A., Vc A., and Ckt B., et al.: An optimal delay aware task assignment scheme for wireless SDN networked edge cloudlets-ScienceDirect. Future Gener. Comput. Syst. 102, 862-875 (2020)

[18] Ahmad S., and Mir A. H.: Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers. J. Netw. Syst. Manag. 29(1) (2021)

[19] Yang F., Cheng L. I., and Huang T.: OXP: An efficient west-east protocol for SDN in Ad hoc. Telecom. Engineering Technics and Standardization. 29(9), 32-37 (2016)

[20] Benamrane F., Mamoun M. B., and Benaini R.: An East-West interface for distributed SDN control plane: Implementation and evaluation. Comput. Electr. Eng. 57, 162-175 (2017)

[21] Yu H., Qi H., and Li K.: WECAN: an Efficient West-East Control Associated Network for Large-Scale SDN Systems. Mobile Netw. Appl. 25(1), 114-124 (2019)

[22] Wu D., Nie X., and Asmare E., et al.: Towards Distributed SDN: Mobility Management and Flow Scheduling in Software Defined Urban IoT. IEEE T. Parall. Distr. 31(6), 1400-1418 (2018)

[23] Wang S., Zhao Y., and Huang L., et al.: QoS prediction for service recommendations in mobile edge computing. J. Parallel Distr. Com. 127, 134-144 (2017)

[24] Xiao Y., Noreikis M., and Yla-Jaaiski A.: QoS-oriented capacity planning for edge computing. In: IEEE International Conference on Communications. IEEE(2017).

[25] Wu B., Zeng J., and Ge L., et al.: Energy-Latency Aware Offloading for Hierarchical Mobile Edge Computing. IEEE Access. 7, 121982-121997 (2019)

[26] Li X.: A computing offloading resource allocation scheme using deep reinforcement learning in mobile edge computing systems. J. Grid Comput. 19(3), 1-12 (2021)

[27] Wang D., Zhao N., and Song B., et al.: Resource Management for Secure Computation Offloading in Softwarized Cyber-Physical Systems. IEEE Internet Things J. 8(11), 9294-9304 (2021)

[28] Ghobaei-Arani M., Souri A., and Rahmanian A. A.: Resource management approaches in fog computing: a comprehensive review. J. Grid Comput. 18(1), 1-42 (2020)

[29] Zhang Y., Fu J.: Energy-efficient computation offloading strategy with tasks scheduling in edge computing. Wirel. Netw. 27(1), 609-620 (2021)

[30] Wang K., Wang X., Liu X.: A High Reliable Computing Offloading Strategy Using Deep Reinforcement Learning for IoVs in Edge Computing. J. Grid. Comput. 19(2) 1-15 (2021)

[31] Yue S., Ren J., and Qiao N., et al.: TODG: Distributed task offloading with delay guarantees for edge computing. IEEE Trans. Parallel Distrib. Syst. 33(7), 1650-1665 (2021)

[32] Sellami B., Hakiri A., and Yahia S. B., et al.: Deep Reinforcement Learning for Energy-Efficient Task Scheduling in SDN-based IoT Network. In: IEEE 19th International Symposium on Network Computing and Applications. IEEE(2020)

[33] Lent R.: A generalized reinforcement learning scheme for random neural networks. Neural Comput. Appl. 31(7), 2699-2716 (2019)

[34] Latif Z., Lee C., and Sharif K., et al.: SDBlockEdge: SDN-Blockchain Enabled Multihop Task Offloading in Collaborative Edge Computing. IEEE Sens. J. 22(15), 15537-15548 (2022)

[35] Volodymyr M., Koray K., and David S., et al.: Human-level control through deep reinforcement learning. Nat. 518(7540), 529-533 (2015)

[36] Huang L., Bi S., and Zhang Y.: Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks. IEEE Trans. Mob. Comput. 19(11), 2581-2593 (2019)

[37] Huang L., Feng X., and Zhang C., et al.: Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. Digital Commun. Networks. 5(1), 10-17 (2019)

[38] Lin P., Bi J., and Wolff S., et al.: A West-East Bridge Based SDN Inter-Domain Testbed. IEEE Commun. Mag. 53(2), 190-197 (2015)

[39] Schaul T., Quan J., Antonoglou I., et al.: Prioritized experience replay. Available Online: arXiv: 1511.05952 (2015)

[40] Sonmez C., Ozgovde A., and Ersoy C.: EdgeCloudSim: An environment for performance evaluation of Edge Computing systems. In: The 2nd International Conference on Fog and Mobile Edge Computing. IEEE(2017)

[41] Bechhoefer E.: The Case of the Missing Bearing Fault Frequency. In: Society for Machinery Failure Prevention Technology conference (2019)

[42] Huang H., Ding S., and Zhao L., et al.: Real-Time Fault-Detection for IIoT Facilities using GBRBM-based DNN. IEEE Internet Things J. 7(7), 5713-5722 (2020)

[43] Zhu A., Wen Y.: Computing Offloading Strategy Using Improved Genetic Algorithm in Mobile Edge Computing System. J. Grid Comput. 19(3), 1-12 (2021)

[44] Feng W., Liu C., and Cheng B., et al.: Secure and cost-effective controller deployment in multi-domain SDN with Baguette. J. Netw. Comput. Appl. 178(2), 102969-102981 (2021)

[45] Hu T., Yi P., and Zhang J., et al.: Reliable and load balance-aware multi-controller deployment in SDN. China Commun. 15(11), 184-198 (2018)

[46] Du R., Liu C., and Gao Y., et al.: Collaborative Cloud-Edge-End Task Offloading in NOMA-Enabled Mobile Edge Computing Using Deep Learning. J. Grid Comput. 20(2), 1-17 (2022)

[47] Kang J., Kim S., and Kim J., et al.: Dynamic Offloading Model for Distributed Collaboration in Edge Computing: A Use Case on Forest Fires Management. Appl. Sci. 10(7), 2334-2347 (2020)

[48] Cho Y. H., and Byun W. J.: Generalized Friis Transmission Equation for Orbital Angular Momentum (OAM) Radios. IEEE Trans. Antennas Propag. 67(4), 2423-2429 (2019)