Computation Offloading and Resource Allocation for Edge Intelligence: A Deep Reinforcement Learning Solution

Xiao Chen, Hongbing Qiu, and Yanlong Li(✉)

Department of Information and Communication, Guilin University of Electronic Technology, Guilin 541004, China
lylong@guet.edu.cn

Abstract. The artificial intelligence training of the terminal clients is mostly limited by the insufficient computing power and energy shortage of the client itself, as well as the client's offline and malfunction, which lead to the terminal intelligent training consuming a large amount of system time. A trusted server is introduced in this scenario, which allows the training tasks of terminal clients to be offloaded to the edge server, to solve this problem. Firstly, a joint optimization model of computation offloading and resource allocation for federated training in mobile edge networks is established. Then the optimization problem is transformed into a Markov process, and the DQN algorithm is applied to obtain the optimal decision. Large amounts of simulation results show that the proposed algorithm reduces the training delay of terminal clients effectively.

Keywords: Artificial Intelligence · Computation Offloading · Resource Allocation · Markov Process

1 Introduction

In recent years, with the widely used of IoT applications, the demand for network bandwidths brought by the transmission of massive data and the increasing real-time demand from mobile users have brought grand challenges to data transmission and data processing of the cloud server. Therefore, the framework that marginalizes computing has come into being. Mobile Edge Computing (MEC) sets up edge servers on the edge of the network to provide computing services for terminal devices and assist clients in completing interactive intelligence training.

With the arising of large amounts of research on edge computing, the optimization strategies of task offloading and resource allocation [1] have become research hotspots as the key to edge computing technology. The joint optimization problems of task offloading and resource allocation are mostly in the form of Mixed Integer Non-Liner Programming (MINLP) with the complexity of NP-hard. A large number of papers solve the problem based on convex optimization theory [2–4]. In addition to the convex optimization theory, some researchers use heuristic algorithms such as genetic algorithm, particle swarm

J. Wang (Ed.): MobiMedia 2023, LNICST 576, pp. 189–198, 2025.
https://doi.org/10.1007/978-3-031-60347-1_16

optimization algorithm, and ant colony algorithm [5] or game theory [6, 7], such as alliance game and Stackelberg theory to optimize the computation offloading strategies in edge computing.

Machine learning which has become very hot in recent years also provides a new solution to this optimization problem. Deep Reinforcement Learning (DRL) integrates the perceptive expression ability of neural networks [8] and the decision ability of reinforcement learning [9] and has received more attention on the joint optimization of computation offloading and resource allocation [10–15].

Computation offloading problems can be divided into local executing, partial offloading, and full offloading. Different from previous data processing tasks, artificial intelligence training generally prefers not to divide the training data. Therefore, training tasks of terminal clients are either executed locally or completely offloaded to the edge server.

At the same time, the artificial intelligence training of the terminal clients is mostly limited by the insufficient computing power and energy shortage of the client itself, as well as the client's offline and malfunction, which lead to the terminal intelligent training consuming a large amount of system time. To alleviate this pressure, this paper proposes a trusted training assistance strategy. A deep reinforcement learning algorithm is used to solve the joint optimization problem of computation offloading and resource allocation. Large amounts of simulation results show that this method is effective and stable in reducing system delay.

The main innovations of this paper are summarized as follows:

(1) A joint optimization model of computation offloading and resource allocation for federated training in mobile edge networks is established.
(2) The optimization problem is transformed into a Markov process, and the optimal decision is solved by using the DQN algorithm. Large amounts of simulation results verify that our algorithm is effective and much more time-saving compared with several baseline algorithms.

The remaining contents are arranged as follows. In Section II, we present the system model. In Section III, a joint optimization algorithm of computation offloading and resource allocation based on DQN is introduced. Section IV presents the experimental results and analyses in detail. At last, we give a conclusion of our work in Section V.

2 System Model

In this paper, a trusted server is considered, which is used to assist the terminal clients' training tasks. The system consists of an edge server and N mobile intelligent clients. Artificial intelligence training can be performed on every terminal client. Orthogonal frequency division multiple access technology is adopted to communicate between the edge server and terminal client.

2.1 Communication Model

The wireless transmission rate between client n and edge server is expressed as Eq. 1.

$$r_n^{up} = B_n \frac{B}{b} log_2(1 + \frac{p_n h_n}{\delta^2}) \tag{1}$$

where B_n denotes the number of unit bandwidth between client n and edge server. B denotes the system bandwidth and b is the total number of unit bandwidth. Every client can occupy serval unit bandwidths, while every unit bandwidth can only be occupied by one client. p_n denotes the transmit power of client n in the upload link. h_n denotes the channel gain at a reference distance d = 1m. δ^2 denotes the noise power. Given the transmission rate and the data size of offloading tasks, the transmission delay can be written as Eq. 2.

$$t_n^{up} = \frac{D_n}{r_n^{up}} \tag{2}$$

where D_n denotes the data size of every training epoch with its unit being bits. Correspondingly, the communication energy consumption can be presented as Eq. 3.

$$e_n^{up} = t_n^{up} \cdot p_n \tag{3}$$

2.2 Computation Model

In our system, training tasks can only be executed locally or all be offloaded to the edge server. When the data generated by client n is offloaded to the edge server, it is processed at the edge server and the calculation time is shown in Eq. 4.

$$t_n^{MEC} = \frac{\varphi_n}{f_n} \tag{4}$$

where f_n denotes the computing capacity that client n allocated from the edge server, its unit is cycles/s. $\varphi_n = D_n \cdot r$, r denotes the number of CPU cycles required to process each bit of data.

When the tasks is executed locally on client n, the calculation time is expressed in Eq. 5.

$$t_n^{IOT} = \frac{\varphi_n}{f_n^{IOT}} \tag{5}$$

where f_n^{IOT} denotes the calculated frequency of client n and the energy consumption of client n can be presented as Eq. 6.

$$e_n^{IOT} = k \cdot \left(\frac{\varphi_n}{T}\right)^2 \cdot \varphi_n \tag{6}$$

where k denotes the effective switching capacitance determined by the chip structure. Therefore, the time consumed on training the data of client n can be presented as Eq. 7.

$$t_n = \alpha_n \cdot \left(t_n^{up} + t_n^{MEC}\right) + (1 - \alpha_n) \cdot t_n^{IOT} \tag{7}$$

When the data of client n executes locally, we set $\alpha_n = 0$, while it is offloaded and trained to the edge server, $\alpha_n = 1$.

Since the edge server can support the entire data processing, we do not think about the energy consumption of the edge server. The energy consumption of client n is defined as Eq. 8.

$$e_n = \alpha_n \cdot e_n^{up} + (1 - \alpha_n) \cdot e_n^{IOT} \tag{8}$$

2.3 Problem Formulation

Based on the communication and computation model, the optimization problem in the trusted server-assisted edge intelligent training system is formulated. To make full use of the limited client energy, computing resources, and bandwidth resources, we optimize task offloading and resource allocation jointly in the system intending to minimize system delay. Specifically, the optimization problem can be expressed as follows:

$$Min_{B_n, f_n, \alpha_n} \sum_{m=1}^{M} \sum_{n=1}^{N} T_n(m) \tag{9a}$$

$$\sum\nolimits_{n=1}^{N} F_{n,m} \leq F^{MEC} m = \{1, 2, \ldots, M\} \tag{9b}$$

$$\sum\nolimits_{n=1}^{N} B_{n,m} \leq Bm = \{1, 2, \ldots, M\} \tag{9c}$$

$$\sum\nolimits_{m=1}^{M} E_{n,m} \leq E_n^{start} n = \{1, 2, \ldots, N\} \tag{9d}$$

$$\alpha_n = \{0, 1\} \tag{9e}$$

where (9a) is our optimization object. There are M steps in each training epoch. Constraint (9b) guarantees that the computing frequencies allocated to all the clients should not be beyond the computation capability of the edge server. Constraint (9c) guarantees that the bandwidth allocated to all the clients should not be beyond system bandwidth. Constraint (9d) guarantees that no client will run out of power in M steps. Constraint (9e) guarantees that the training task can only be executed locally or be offloaded fully to the edge server.

3 DQN-Based Computation Offloading and Resource Allocation Optimization

Deep reinforcement learning combines deep learning and reinforcement learning. It uses deep neural networks to identify high-dimensional data effectively, making reinforcement learning algorithms more effective in processing high-dimensional state space tasks. In this paper, we convert the optimization problem into a Markov decision problem. Firstly, we need to determine the state space, action space, and reward function.

3.1 State Space

In the mth step, the state of the agent contains the basic information of all clients, the current resource allocation and offloading condition. The state of the agent can be expressed as Eq. 10.

$$S_m = \{e_{n,m}, B_{n,m}, F_{n,m}, flag_{n,m}\} \tag{10}$$

where $n \in \{1,2,3...,N\}$, $e_{n,m}$ denotes the remaining energy of client n in the mth step. $B_{n,m}$ Denotes the allocated bandwidth in this system of client n in the mth step. $F_{n,m}$ Denotes the allocated computation capability of client n in the mth step from the edge server. $flag_{n,m}$ Denotes the flag of data offloading of client n in the mth step.

3.2 Action Space

In the mth step, the agent action includes the decision of computation offloading and the decision of resource allocation. The agent action can be expressed as Eq. 11.

$$A_m = \{\Delta B_{n,m}, \Delta F_{n,m}, \Delta flag_{n,m}\} \tag{11}$$

where $n \in \{1,2,3...,N\}$, $\Delta B_{n,m}$ denotes the variation of the bandwidth allocated to client n in the mth step. $\Delta F_{n,m}$ denotes the variation of the computation capability allocated to client n in the mth step. $\Delta flag_{n,m}$ denotes the flag of task offloading of client n in the mth step.

3.3 Reward Function

For the agent, when performing each action in a certain state, we can obtain a corresponding reward value, which will guide the learning direction of the agent. The agent will enter the t + 1 after performing A_t from the state of S_t. According to A_t and S_{t+1}, the corresponding reward value R_{t+1} can be obtained. In general, the reward function is related to the optimization goal. In this scenario, our objective of the joint optimization of computation offloading and resource allocation is to minimize the model training delay, while the target of reinforcement learning is to maximize the long-term cumulative reward. The reward function in the mth step is defined as Eq. 12.

$$R_t = \frac{1}{max\{T_n\}} \tag{12}$$

where $T_n = \alpha_n \bullet (t_n^{up} + t_n^{MEC}) + (1 - \alpha_n) \bullet t_n^{IOT}$, $n \in \{1,2,3,\ldots,N\}$.

3.4 Dqn

DQN algorithm calculates Q value based on Q-learning and uses a neural network to simulate Q function. After we gather sufficient training samples, the samples in the cache pool are used to train the neural network parameters of the Q function, and the neural network gradually approaches the actual Q function after continuous iterations.

The algorithm adopts the approximate action function Q (s, a, w) output by two fully connected layers (32 nodes in the first hidden layer, 64 nodes in the second hidden layer) to approximate the real action-value function $q_\pi(s, a)$.

$$Q(s, a, w) \approx q_\pi(s, a) \tag{13}$$

In general, we determine the loss function $L(w)$ first, then calculate its gradient and update the parameter w by gradient descent or other optimization methods. DQN constructs the loss function of the network based on the Q-Learning algorithm.

$$L(w) = E_{\pi_w}\left[\left(r + \gamma max_{a'} Q\left(s', a', w\right) - Q(s, a, w)\right)^2\right] \tag{14}$$

where w is the network parameters, $r + \gamma max_{a'} Q\left(s', a',\ w\right)$ is the target Q value, and Q (s, a, w) is the predicted Q value.

Target Network. DQN calculates the target Q value using a dedicated target network instead of the pre-updated Q network. Since the Q value of the target network remains Unchanged for a while, the correlation between the Predicted Q value and the target Q value is reduced to a certain extent, and the possibility of loss oscillation and divergence is reduced during training, Which fully guarantees the training time and improves the stability of the algorithm. The formula for calculating the DQN loss function $L(w)$ under dual network architecture is Defined as Eq. 15.

$$L(w) = E_{\pi_w}[(r + \gamma max_{a'} Q(s\prime, a\prime, w\prime) - Q(s, a, w))^2] \tag{15}$$

Experience Replay. The input samples are required to be independent and equally distributed from each other in deep learning, while samples are often associated and non-static in reinforcement learning. If we train the model with the associated data directly, problems such as the difficulty of convergence and continuous fluctuation of the loss value will be caused.

There goes the detailed process of the DQN algorithm in algorithm 1.

Initialize training episodes M.
Initialize the value of replay memory D to N.
Initialize the parameter w of prediction network and the parameter w' of target network with random weights.
Initializes the update frequency of the target network parameters C.

For episode=1 to M do

Initialize sequence $S_1=\{x_1\}$ and get the preprocessed sequence $\phi_1 = \phi(s_1)$.

Input ϕ_1 to the prediction network and get the Q function corresponding to every action:$Q(\phi(s_1), a, w)$.

Repeat (each time slot t)

Greedy strategy is used to select actions in the prediction network： $A_t = arg\ max_a Q(\phi(s_1), a, w)$.

Execute action A_t and observe reward R_{t+1} and the next state x_{t+1}.

Set $S_{t+1} = S_t$， and get the preprocessed sequence $\phi_{t+1} = \phi(s_{t+1})$.

Store $(\phi_t, A_t, R_{t+1}, \phi_{t+1})$ in D.

Batchsize-n empirical transfer samples are randomly selected from D when the samples in replay memory are much larger than *batchsize-n:*

$[(\phi_1, A_1, R_2, \phi_2),\ldots,(\phi_i, A_i, R_{i+1}, \phi_{i+1}),\ldots, (\phi_n, A_n, R_{n+1}, \phi_{n+1})]$

Set $y_i = \begin{cases} R_{i+1} & for\ terminal\ \phi_{i+1} \\ R_{i+1} + \gamma max_a Q(\phi_{i+1,a,w'}) & otherwise \end{cases}$

Calculate the loss function$(y_i - Q(\phi_i, A_i, w))^2$ and update the parameter of prediction network with MBSGD algorithm.

Until t is terminated.

Update the parameter of target network $w' = w$ every C steps.

End for

4 Results and Analysis

4.1 Simulation Setting

In this section, the effectiveness of the proposed DQN-based resource allocation and computation offloading scheme is verified. The simulation environment is Python 3.8.8 and Pytorch 1.9.1. In the experiment, the amount of data to be trained for each step is set as 200Mbits, and the computation capability of the edge server is set as 40GHz. The other parameters are listed in Table 1.

To verify the effectiveness of our algorithm, we compare it with the other three baseline algorithms.

Random. Whether the samples of each client are offloaded, the amount of the allocated system bandwidth and the amount of allocated computation capability from the edge server are determined randomly.

Table 1. Table captions should be placed above the tables.

Symbols	Means	Values
K	the number of clients	3
b	the number of unit bandwidth	8
r	the number of CPU cycles required to process each bit of data	1000
h	the channel gain at a reference distance $d = 1$m	-30dB
p	uplink transmission power	1W
δ^2	noise power	-100dBm
s	the influence factor of chip structure on CPU processing	10^{-25}
F	the computing frequencies of the edge server	40GHz

Local-Only. Samples of all clients train locally.

Offload-Only. Samples of all clients train in the edge server.

4.2 Simulation Results and Discussions

Parametric Analysis. Firstly, we conduct large amounts of experiments to explore the optimal hyperparameters in our resource allocation and computation offloading scheme based on the DQN Algorithm. The convergence curves of the proposed algorithm under different learning rates are shown in error! reference source not found. When the learning rate is 0.001, It is difficult to converge with the optimization process oscillating back and forth next to the optimal value. We obtain the convergence curves in the other two cases, but when the learning rate is 0.00001, the convergence rate is slower than that with the learning Rate is 0.0001 (Fig. 1).

Performance comparison. Figure 2 Shows the curves of system delay from four different computation offloading and resource allocation schemes. As shown in Fig. 2, the delay of local-only is fixed. Due to the limitation of system transmission bandwidth or the computation capability supplied by the edge server, the delay of the Offload-only scheme is large. And with the increase of system resources, the delay of Offload-only scheme will decrease significantly. The system delay of the random scheme is distributed randomly. The scheme based on DQN has strong learning and decision-making ability, and its curve tends to converge after training hundreds of epochs. Meanwhile, the minimum system delay can be obtained by the DQN algorithm compared with other algorithms.

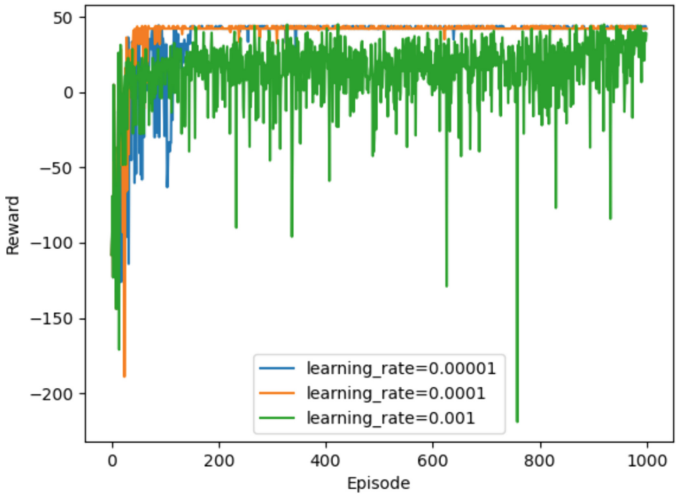

Fig. 1. Curves under different learning rates.

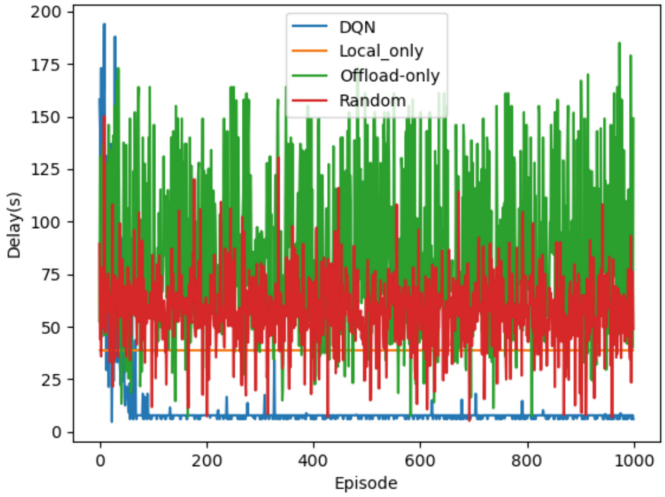

Fig. 2. Comparison of different algorithm.

5 Conclusion

In this paper, a trusted server is introduced in the edge intelligent computing scenario, which allows the training tasks of terminal clients to be offloaded to the edge server, to solve the problem of limited energy and computation capability of terminal clients. Firstly, a joint optimization model of computation offloading and resource allocation for federated training in mobile edge networks is established. Then the optimization problem is transformed into a Markov process, and the DQN algorithm is applied to obtain the optimal decision. Simulation results show that the proposed algorithm effectively

reduces the training delay of terminal clients. More effort will be put into other deep reinforcement learning algorithms in the next step, to solve the computation offloading and resource allocation.

Funding. This work is supported by the National Natural Science Foundation of China under grant 62261009, Innovation Project of Guangxi Graduate Education (YCBZ2022106) and the Open Funds from Guilin University of Electronic Technology (GIIP2006).

References

1. Mach, P., Becvar, Z.: Mobile edge computing: A survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. **19**(3), 1628–1656 (2017)
2. Sardellitti, S., Scutari, G., Barbarossa, S.: Joint optimization of radio and computational resources for multicell mobile-edge computing. IEEE Trans. Signal Inf. Process. Over Netw. **1**(2), 89–103 (2015)
3. Li, Y., Liang, L., Fu, J., Wang, J.: Multiagent reinforcement learning for task offloading of space/aerial-assisted edge computing. Sec. Commun. Netw. **2022**, 4193365 (2022)
4. Yu, Y., Bu, X., Yang, K., et al.: UAV-aided low latency mobile edge computing with mmWave backhaul. In: ICC 2019 – 2019 IEEE International Conference on Communications (ICC), pp. 1–7. IEEE, Shanghai, China, (2019)
5. Wang, Y., Fang, W., Ding, Y., et al.: Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach. Wireless Netw. **27**, 2991–3006 (2021)
6. Zhang, K., Gui, X., Ren, D., et al.: Energy-latency tradeoff for computation offloading in UAV-assisted multiaccess edge computing system. IEEE Internet Things J. **8**(8), 6709–6719 (2020)
7. Liu, J. et al.: Minimization of offloading delay for two-tier UAV with mobile edge computing. In: 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), pp. 1534–1538. Tangier, Morocco, (2019)
8. Lv, J., et al. Joint Computation Offloading and Resource Configuration in Ultra-Dense Edge Computing Networks: A Deep Reinforcement Learning Solution. In: *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pp. 1–5. Honolulu, HI, USA, (2019)
9. Mnih, V. et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
10. Pan, S., et al.: Dependency-aware computation offloading in mobile edge computing: a reinforcement learning approach. IEEE Access **7**, 134742–134753 (2019)
11. Li, J., et al.: Deep reinforcement learning based computation offloading and resource allocation for MEC. In: 2018 IEEE Wireless Communications and Networking Conference (WCNC), pp.1–6. IEEE, Barcelona, Spain (2018)
12. Chen, X., et al.: Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J. **6**(3), 4005–4018 (2018)
13. Huang, L., Bi, S., Zhang, Y.-J.A.: Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans. Mob. Comput. **19**(11), 2581–2593 (2020)
14. Zhou, C., et al.: Delay-aware IoT task scheduling in space-air-ground integrated network. In: 2019 IEEE Global Communications Conference (GLOBECOM). pp. 1–6. IEEE, Waikoloa, HI, USA (2019)
15. Zhou, C., et al.: Deep reinforcement learning for delay-oriented IoT task scheduling in SAGIN. IEEE Trans. Wireless Commun. **20**(2), 911–925 (2020)