

RESEARCH

Open Access



Joint DNN partitioning and task offloading in mobile edge computing via deep reinforcement learning

Jianbing Zhang¹, Shufang Ma¹, Zexiao Yan¹ and Jiwei Huang^{1*}

Abstract

As Artificial Intelligence (AI) becomes increasingly prevalent, Deep Neural Networks (DNNs) have become a crucial tool for developing and advancing AI applications. Considering limited computing and energy resources on mobile devices (MDs), it is a challenge to perform compute-intensive DNN tasks on MDs. To attack this challenge, mobile edge computing (MEC) provides a viable solution through DNN partitioning and task offloading. However, as the communication conditions between different devices change over time, DNN partitioning on different devices must also change synchronously. This is a dynamic process, which aggravates the complexity of DNN partitioning. In this paper, we delve into the issue of jointly optimizing energy and delay for DNN partitioning and task offloading in a dynamic MEC scenario where each MD and the server adopt the pre-trained DNNs for task inference. Taking advantage of the characteristics of DNN, we first propose a strategy for layered partitioning of DNN tasks to divide the task of each MD into subtasks that can be either processed on the MD or offloaded to the server for computation. Then, we formulate the trade-off between energy and delay as a joint optimization problem, which is further represented as a Markov decision process (MDP). To solve this, we design a DNN partitioning and task offloading (DPTO) algorithm utilizing deep reinforcement learning (DRL), which enables MDs to make optimal offloading decisions. Finally, experimental results demonstrate that our algorithm outperforms existing non-DRL and DRL algorithms with respect to processing delay and energy consumption, and can be applied to different DNN types.

Keywords Deep neural networks(DNNs), Mobile edge computing(MEC), DNN partitioning, Task offloading, Deep reinforcement learning (DRL)

Introduction

As a core technology supporting modern Artificial Intelligence (AI) mobile applications, Deep Neural Networks (DNNs) have widespread applications in computer vision, natural language processing, image recognition, virtual reality (VR), augmented reality (AR) and other fields [1–3]. However, considering that the high computational complexity of DNN-based inference tasks, it is difficult

to execute these DNN-based inference tasks directly on mobile devices (MDs) having constrained computation and energy resources.

In order to cope with the excessive demand on computing resources for compute-intensive DNN tasks, the traditional solution resorts to the cloud datacenter with strong computing power for intensive computation [4]. In this case, the task data arriving at MDs is transmitted to the remote cloud datacenter for computation, and the result is returned to the local devices once the computation is complete. However, this cloud-based method involves the transmission of large amounts of data via a long-distance wide-area network (WAN), resulting in high transmission energy consumption and delay, and

*Correspondence:

Jiwei Huang
huangjw@cup.edu.cn

¹ Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum, Beijing, China

cannot meet the requirements of energy-sensitive and delay-sensitive DNN inference tasks. To address this issue, mobile edge computing (MEC) [5–7] is proposed as an emergent computing mode that places computational and storage resources on edge nodes near MDs [8], enabling compute-intensive DNN-based applications to be executed in a real-time responsive manner, i.e., edge intelligence [1, 9, 10]. This novel manner can meet the low delay and low energy consumption requirements of DNN tasks [11–14]. In MEC, we can take advantage of the characteristics of DNN to offload part or all of the tasks on MDs to the MEC server to support real-time edge AI applications [15, 16].

Although edge intelligence technology has brought many benefits [17], edge-based DNN inference tasks remain heavily reliant on stable and reliable communication conditions between the MD and the edge server [9]. Considering that the network environment is often changing and easily disturbed in actual deployment, it is particularly important to further optimize DNN partitioning and task offloading in the dynamic network environment. Therefore, in response to the ever-changing network environment, we need to explore more robust and flexible DNN partitioning and task offloading techniques. In addition, since the number of tasks generated by MDs is randomly variable, the probability of the state transition is unknown.

On these issues mentioned above, in this paper, we first dynamically partition DNN tasks by layer into subtasks that can be executed on the MD or the MEC server [18]. Towards low delay and low energy consumption edge intelligence, we utilize the joint optimization method to formulate the energy-delay trade-off of MDs as a Markov decision process (MDP) [19]. Thereafter, we adopt traditional deep reinforcement learning (DRL) algorithms, including the Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) algorithms, to make MDs learn the optimal offloading policy while considering the future dynamic characteristics of the system environment. Nevertheless, we find that traditional DRL algorithms do not converge or converge slowly. To address this problem, we design a DNN partitioning and task offloading strategy based on Proximal Policy Optimization (PPO) algorithm, which can decrease energy consumption, reduce processing delay, and can also be extended to various types of DNNs. In the end, numerous simulation experiments are carried out to validate the effectiveness of our proposed method in enabling on-demand edge intelligence with low delay and low energy consumption.

In summary, this paper makes the following main contributions:

- We present a novel approach for DNN task partitioning, which utilizes a layered partitioning method to

divide the tasks of each MD into smaller subtasks that can be computed on the MD or offloaded to the server for processing.

- We study the optimization of energy consumption and processing delay for DNN partitioning and task offloading in a dynamic MEC scenario consisting of multiple MDs with buffer and one MEC server, where each MD and the MEC server use the pre-trained DNNs for task inference. And we construct the processing delay model and energy consumption model for the MEC system. Then, we further formulate the optimization problem as an MDP problem.

- To address the above issue, we propose a DNN partitioning and task offloading (DPTO) algorithm based on DRL. At the same time, we evaluate the processing delay, energy consumption, and utility of our DPTO algorithm in simulation experiments, and numerous experimental results indicate that our DPTO algorithm surpasses existing non-DRL and DRL algorithms, effectively reduces the processing delay and energy consumption, and simultaneously can be applied to various types of DNNs.

The rest of this paper is structured as follows. Section “[Related work](#)” presents a detailed overview of the related work that is most relevant to this paper. In section “[System model and problem formulation](#)”, we first introduce the processing delay model and the energy consumption model, and then expound the process of modelling the joint optimization problem as an MDP problem. Section “[DRL-based algorithm design](#)” discusses the design of our DPTO algorithm based on DRL. In section “[Performance evaluation](#)”, we conduct extensive simulation experiments to evaluate the performance of our proposed approach. Finally, in section “[Conclusion](#)”, we summarize our contributions and conclude the paper.

Related work

Recently, discussions on DNN partitioning and task offloading have received more and more attention. Since the number of data generated by the computation of some intermediate layers of the DNN model is relatively small, they are sent to the edge server with less transmission energy consumption and delay than the original data through the network, which stimulates the method of DNN partitioning and task offloading [1]. In addition, given the multi-layer structure of the DNN and the strong interdependence between neurons in each layer, it is difficult to partition computations in the same layer. And since there are some restrictions on the granularity of DNN partitioning in programming, it is not feasible to partition DNN arbitrarily. Therefore, designing an effective DNN partitioning and task offloading scheme is a challenging problem.

To cope with this challenge, many researchers have made some efforts in the field of DNN partitioning and task offloading. For example, a lightweight scheduler, i.e., Neurosurgeon, tailored for a basic edge computing network consisting of a single user and one server was introduced by Kang et al. [18]. The scheduler facilitated automatic DNN partitioning between the MD and the datacenter. However, applying Neurosurgeon to complex multi-user MEC networks presents some unresolved issues. He et al. [20] assumed that a fixed set of partitions was used to partition the DNN. The main focus of their approach was to optimize the DNN partitioning on the MEC server to minimize delay, instead of selecting partition points. Nevertheless, the fixed partition deployment of DNN on the MEC server may not be practical in multi-user MEC networks due to the various types of DNNs. In addition, Gao et al. [14] introduced two novel approaches that aim to optimize the partitioning and offloading of DNN tasks. And these two algorithms have the best performance in terms of delay, energy, and the price paid to the server for each MD. Furthermore, they are also extended to a wide range of DNN types.

According to the above DNN partitioning and task offloading strategy, for MDs with constrained computing and energy resources, task offloading to servers for processing is a feasible solution [21–23]. Some existing work focuses on the delay optimization in task offloading. Specifically, Li et al. [9] introduced a framework called Edgent for collaborative inference of DNN utilizing edge computing through device-edge synergy. Their approach facilitated adaptive partitioning of DNN computation between the device and edge, and allowed for premature termination of inference at a suitable intermediate DNN layer to minimize computation delay. To increase the amount of allowable delay-aware DNN service requests, Li et al. [24] devised a new strategy that involves optimizing DNN partitioning and multi-thread execution parallelism. This approach aims to maximize the throughput of DNN inference, which is especially crucial in DNN-based applications that require real-time processing. Chen et al. [4] proposed a solution to the problem of excessive delay in offloading by delegating partially compute-intensive tasks to remote clouds or edges. The latest researches [25, 26] investigated task offloading policies aimed at fulfilling the low-latency demands of users.

Another area of related work focuses on energy optimization in offloading. For example, Chen et al. [27] addressed the problem of dynamic task offloading in digital twin-enabled MEC, and designed an energy-efficient algorithm based on DRL with the goal of maximizing energy efficiency and workload balancing among the ESs. In [28], they proposed a solution that utilizes DRL to tackle the challenge of AOI-aware energy control and

computing offloading in a dynamic IIoT environment. The approach designed by the authors enables effective energy management and computing offloading, while considering the changing nature of the IIoT system. Li et al. [29] developed an energy-efficient algorithm to minimize total energy consumption. The trade-off between system performance and energy consumption was investigated by Zhou et al. [30] in a multi-cloud system using UAVs.

The third type of work relates to the joint optimization of delay and energy. To be concrete, in the research on computing offloading for IOT devices in LEO satellite edge computing, Chen et al. [31] investigated the challenge of ensuring Qos while minimizing overall costs. To address this issue, they proposed a distributed approach that takes into account multiple constraints, including computing resources, delay and energy consumption, to achieve Qos-aware computing offloading. In [32], the author investigated the energy and delay trade-off in a MEC system using energy harvesting devices. Furthermore, there have been studies [33, 34] that have proposed an online Lyapunov optimization technique for balancing the energy consumption and delay.

Although there are numerous studies on jointly optimizing energy and delay in task offloading, these existing offloading methods are not common for offloading DNN tasks in dynamic MEC systems. Therefore, unlike previous works, this paper primarily concentrates on effectively tackling the challenge of reducing the energy consumption and processing delay in a dynamic MEC scenario through DNN partitioning and task offloading. To address this issue, we put forward a DRL-based DPTO approach, which is detailed in Section “DRL-based algorithm design”.

System model and problem formulation

Our research focuses on a dynamic MEC scenario, which involves a plurality of MDs embedded with task buffers for temporarily storing unprocessed tasks and one MEC server, as illustrated in Fig. 1. We use $U = \{1, 2, \dots, n\}$ to represent the collection of MDs, where n is an integer representing the overall amount of MDs in the collection. And each MD and the MEC server in this MEC scenario use pre-trained DNNs to compute their tasks. We consider a system with time slots, each with length τ . The time slots are indexed by $t \in \{0, 1, \dots, T - 1\}$. The u th MD receives $D_u(t)$ DNN tasks and has $Q_u(t)$ tasks currently stored in the buffer at the start of time slot t . The sum of $D_u(t)$ and $Q_u(t)$ gives the total size of tasks that the u th MD needs to process. In addition, the DNN model adopted by the u th MD has L_u layers.

Figure 1 provides a detailed illustration of DNN partitioning and task offloading for MDs. The u th MD has the

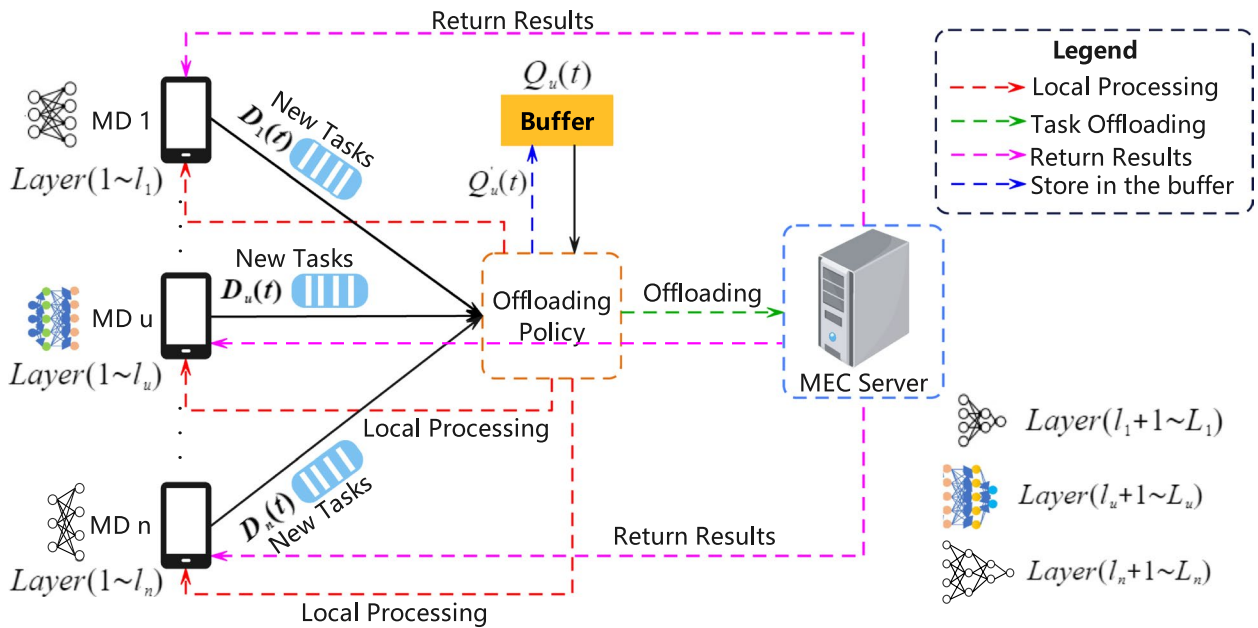


Fig. 1 System Model

flexibility to choose between two task execution modes: local computation and offloading to the server via a reliable wireless channel for processing. According to the offloading policy, it can further decide how many layers to compute on the MD and how many layers to compute on the MEC server, both of which are indicated by $\alpha_{u,L}(t)$ and $\alpha_{u,M}(t)$. Specifically, when $\alpha_{u,L}(t) = L_u, \alpha_{u,M}(t) = 0$, means that the tasks are computed locally on the MD, but when $\alpha_{u,L}(t) = 0, \alpha_{u,M}(t) = L_u$, means that the tasks are computed on the MEC server. Specially, when $\alpha_{u,L}(t)$ and $\alpha_{u,M}(t)$ are both 0, this indicates that the tasks are stored in the buffer temporarily for later processing, instead of being computed. The unexecuted tasks are represented by $Q'_u(t)$, which can be modeled as $Q'_u(t) = (D_u(t) + Q_u(t)) \cdot \left(1 - \frac{\alpha_{u,L}(t) + \alpha_{u,M}(t)}{L_u}\right)$. In brief, the decision made by the u th MD could be considered as an action tuple $a_u = [\alpha_{u,L}(t), \alpha_{u,M}(t)] \in A$, in which A denotes the collection of all the action tuples and $\alpha_{u,L}(t) + \alpha_{u,M}(t) \in \{0, L_u\}$. The system model uses the primary notations listed in Table 1.

Thereafter, we will provide a detailed system model by analyzing two aspects, including processing delay model and energy consumption model.

Processing delay model

Since the delay caused by task partitioning is very small compared to the total delay during the entire DNN task processing, we can disregard it. Therefore, the processing delay consists of four parts: the computation delay of the u th MD $T_{u,loc}(t)$, the data upload delay from the

u th MD to the server $T_{u,up}(t)$, the computation delay of the server $T_{u,mec}(t)$, and the data download delay from the server to the u th MD $T_{u,do}(t)$. We partition DNN tasks on the u th MD by layer into subtasks, which are represented by the sequence $\{M_{u,1}, \dots, M_{u,l}, \dots, M_{u,L_u}\}$, where $1 \leq l \leq L_u$. Denote $M_{u,l}$ as the subtask l on the u th MD. The input matrix size of each layer in the DNN model depends on the specific model structure and the dimensions of the input data. In general, for convolutional layers, the input matrix size depends on the size of the input image, the number of channels, and the size of the convolutional kernel. For fully connected layers, the input matrix size is determined by the number of output nodes in the previous layer and the number of nodes in the current layer. With reference to [35], for the inference request on the u th MD, the input matrix size is in direct proportion to the computation delay of each layer in the DNN. We call $f_{u,l}$ the ratio of the input matrix of the DNN layer l on the u th MD to the initial data size $D_u(t)$. Specially, $f_{u,1} = 1$. For the subtask $M_{u,l}$, we denote the input matrix size by $Y_{u,l}(t)$ and it is represented as follows:

$$Y_{u,l}(t) = f_{u,l} \cdot D_u(t), \tag{1}$$

where the value of $f_{u,l}$ can be derived from the historical data on inference requests during the model training process.

With reference to [36–38], we can represent the computation delay $T_{u,l}^{loc}(t)$ for the DNN layer l on the u th MD as

Table 1 Notations and definitions

Notation	Definition
n	The number of MDs
$D_u(t)$	The task size on the u th MD in time slot t
$Q_u(t)$	The task size currently stored in the buffer of the u th MD in time slot t
L_u	The amount of layers of the DNN model on the u th MD
$\alpha_{u,l}(t)$	The amount of layers computed on the u th MD in time slot t
$\alpha_{u,M}(t)$	The amount of layers computed on the MEC server in time slot t
$Q'_u(t)$	The unexecuted task size on the u th MD in time slot t
A	The collection of all the action tuples
$M_{u,l}$	The subtask l on the u th MD
$f_{u,l}$	The ratio of input matrix of the DNN layer l on the u th MD to the initial data size $D_u(t)$
$Y_{u,l}(t)$	The input matrix size of the subtask $M_{u,l}$ in time slot t
$\xi_{u,loc}(t)$	The processing time taken by the u th MD to process one unit of data in time slot t
$\xi_{u,loc}^{min}$	The minimum processing time taken by the u th MD to process one unit of data
$\xi_{mec}(t)$	The time taken by the server to process one unit of data in time slot t
ξ_{mec}^{min}	The minimum time taken by the server to process one unit of data
$B_u(t)$	The bandwidth between the u th MD and the MEC server in time slot t
N_0	The power spectral density of noise
$h_u(t)$	The channel power gain between the u th MD and the MEC server in time slot t
$P_u^{up}(t)$	The uploading power of the u th MD in time slot t
$P_M(t)$	The transmission power of the server in time slot t
$O_{k_u}(t)$	The output data size from the DNN layer k_u on the u th MD in time slot t
$O_{L_u}(t)$	The output data size from the last DNN layer L_u on the u th MD in time slot t
$P_u^{exe}(t)$	The computing power of the u th MD in time slot t
S	The collection of all the states
$C_u(t)$	The overall cost for the u th MD in time slot t
$R_u(t)$	The reward value corresponding to $(s_u(t), a_u(t))$

$$T_{u,l}^{loc}(t) = Y_{u,l}(t) \cdot \xi_{u,loc}(t), \quad (2)$$

where $\xi_{u,loc}(t)$ represents the processing time taken by the u th MD to process one unit of data, and $Y_{u,l}(t)$ is given in (1). Similarly, the computation delay $T_{u,l}^{mec}(t)$ for the DNN layer l of the u th MD on the MEC server as

$$T_{u,l}^{mec}(t) = Y_{u,l}(t) \cdot \xi_{mec}(t), \quad (3)$$

where $\xi_{mec}(t)$ is a constant representing the time required by the MEC server to process one unit of data. We suppose that the DNN layer 1 to layer k_u is computed on the u th MD, and layer $k_u + 1$ to layer L_u is computed on the server, where k_u represents the DNN layer k on the u th MD, and $1 \leq k_u \leq L_u$. Therefore, we can represent the computing delay on the u th MD locally as

$$T_{u,loc}(t) = \sum_{l=1}^{k_u} T_{u,l}^{loc}(t), \quad (4)$$

where $T_{u,l}^{loc}(t)$ is given in (2). We can model the computing delay on the MEC server as follows:

$$T_{u,mec}(t) = \sum_{l=k_u+1}^{L_u} T_{u,l}^{mec}(t), \quad (5)$$

where $T_{u,l}^{mec}(t)$ could be obtained in (3).

We consider that MDs could send the processed data to the MEC server, causing additional transmission delay and energy consumption during the transmission period. Specifically, based on the Shannon theory [39], we can denote the transmission data rate $R_{u,up}(t)$ from the u th MD to the MEC server as

$$R_{u,up}(t) = B_u(t) \log_2 \left(1 + \frac{h_u(t) P_u^{up}(t)}{B_u(t) N_0} \right), \quad (6)$$

where $B_u(t)$ and $h_u(t)$ are the channel bandwidth and channel power gain available between the u th MD and the MEC server, respectively, N_0 represents the power spectral density of noise, and the u th MD has the upload power $P_u^{up}(t)$. In addition, we use $O_{k_u}(t)$ to denote the output data size from the DNN layer k_u on the u th MD

in time slot t . Thereafter, according to (6), the data upload delay from the u th MD to the MEC server is

$$T_{u,up}(t) = \frac{O_{k_u}(t)}{R_{u,up}(t)}. \quad (7)$$

Similarly, during the download of the output results computed by the MEC server to the MDs, there will also produce transmission delay. We use $R_{u,do}(t)$ to indicate the data download rate from the MEC server to the u th MD, which is given by:

$$R_{u,do}(t) = B_u(t) \log_2 \left(1 + \frac{h_u(t)P_M(t)}{B_u(t)N_0} \right), \quad (8)$$

where the transmission power of the MEC server is denoted by $P_M(t)$. According to (8), we can obtain the data download delay from the MEC server to the u th MD $T_{u,do}(t)$ as

$$T_{u,do}(t) = \frac{O_{L_u}(t)}{R_{u,do}(t)}, \quad (9)$$

where the output data size from the last DNN layer L_u on the u th MD in time slot t is represented by $O_{L_u}(t)$.

Specially, when the tasks are computed locally, i.e., $\alpha_{u,L}(t) = L_u$, $\alpha_{u,M}(t) = 0$, we do not upload data to the MEC server, correspondingly, the download delay is zero, and so is the computation delay of the MEC server. On the contrary, if $\alpha_{u,L}(t) = 0$, $\alpha_{u,M}(t) = L_u$, i.e., the tasks are computed only on the server. Therefore, the local computation delay on the u th MD is zero. If $\alpha_{u,L}(t) = 0$, $\alpha_{u,M}(t) = 0$, i.e., the tasks need to wait for later processing. In the current time slot t , the processing delay and energy consumption of the u th MD are both equal to 0. To sum up, the total processing delay of the u th MD can be modeled as

$$T_u(t) = \begin{cases} 0, & \alpha_{u,L}(t) = 0, \alpha_{u,M}(t) = 0 \\ T_{u,loc}(t), & \alpha_{u,L}(t) = L_u, \alpha_{u,M}(t) = 0 \\ T_{u,M_1}(t), & \alpha_{u,L}(t) = 0, \alpha_{u,M}(t) = L_u \\ T_{u,M_2}(t), & other, \end{cases} \quad (10)$$

where $T_{u,M_1}(t) = T_{u,mec}(t) + T_{u,up}(t) + T_{u,do}(t)$ and $T_{u,M_2}(t) = T_{u,loc}(t) + T_{u,mec}(t) + T_{u,up}(t) + T_{u,do}(t)$.

Energy consumption model

The MEC server has an uninterrupted power supply from the power grid, so we can disregard the energy consumption of the MEC server for computing and downloading the calculation results to the MDs [19]. Furthermore, the energy consumption caused by task partitioning is so small for the entire DNN task processing process that we can ignore it. Therefore, the main energy consumption in our system comes from the computing energy of MDs and the data transmission energy from MDs to the MEC

server. We can model the computing energy consumption of DNN tasks on the u th MD as

$$E_{u,loc}(t) = P_u^{exe}(t)T_{u,loc}(t), \quad (11)$$

where the computing power of the u th MD is denoted by $P_u^{exe}(t)$. $T_{u,loc}(t)$ is the computing delay on the u th MD and is given in (4). The energy consumption of uploading the output data of the DNN layer k_u executed by the u th MD to the MEC server is

$$E_{u,up}(t) = P_u^{up}(t) \cdot \min\{T_{u,up}(t), \tau\}, \quad (12)$$

where τ is the duration of each slot.

Specially, when the tasks are computed only on the MD, the MD does not need to send data to the MEC server. Therefore, the only energy cost for the u th MD is the local computing energy cost. On the contrary, if the tasks are computed only on the MEC server, the only energy consumption $E_u(t)$ is the energy consumed to send data from the u th MD to the server. Overall, the energy consumption $E_u(t)$ for the u th MD is given by the following formula:

$$E_u(t) = \begin{cases} 0, & \alpha_{u,L}(t) = 0, \alpha_{u,M}(t) = 0 \\ E_{u,loc}(t), & \alpha_{u,L}(t) = L_u, \alpha_{u,M}(t) = 0 \\ E_{u,up}(t), & \alpha_{u,L}(t) = 0, \alpha_{u,M}(t) = L_u \\ E_{u,M}(t), & other, \end{cases} \quad (13)$$

where $E_{u,M}(t) = E_{u,loc}(t) + E_{u,up}(t)$.

Problem formulation

According to the conditions of the current channel, MDs decide the amount of DNN layers to be computed on the MEC server and locally. When the channel conditions remain good for an extended period, the amount of tasks in the buffer $Q_u(t)$ will become zero after a time slot $t > 0$. At this time, the number of tasks arriving at the u th MD $D_u(t)$ is equal to the number of tasks to be processed.

Storing DNN tasks in the buffer temporarily will have a substantial influence on the service quality and user satisfaction. Thus, we denote $\omega \cdot Q'_u(t)$ as the punishment. In summary, in time slot t , the total cost of the u th MD consists of processing delay, energy cost, and punishment, and it can be modeled as

$$C_u(t) = \mu \cdot T_u(t) + \nu \cdot E_u(t) + \omega \cdot Q'_u(t), \quad (14)$$

where μ , ν , ω are the weights of delay, energy cost, and punishment respectively.

The objective is to finish the DNN tasks quickly and use the least amount of energy, by reducing the system cost as much as possible without exceeding the computational limits. We define $\xi_{u,loc}^{min}$ and ξ_{mec}^{min} as the minimum time taken by the u th MD and the server to process a unit

of data, respectively. $R_{u,up}^{max}$ is the maximum data upload speed from the u th MD to the server. $R_{u,do}^{max}$ is the maximum data download speed from the server to the u th MD. In summary, the following expression shows how to describe the problem:

$$\mathbf{P1} : \min \lim_{T \rightarrow \infty} \frac{\sum_{t=0}^{T-1} \sum_{u=1}^n C_u(t)}{n}, \quad (15)$$

$$s.t. \quad \alpha_{u,L}(t) + \alpha_{u,M}(t) \in \{0, L_u\}, \forall u \in U, \quad (16)$$

$$\xi_{u,loc}(t) \geq \xi_{u,loc}^{min}, \forall u \in U, \quad (17)$$

$$\xi_{mec}(t) \geq \xi_{mec}^{min}, \quad (18)$$

$$0 \leq R_{u,up}(t) \leq R_{u,up}^{max}, \forall u \in U, \quad (19)$$

$$0 \leq R_{u,do}(t) \leq R_{u,do}^{max}, \forall u \in U, \quad (20)$$

where the constraint (16) indicates that the tasks are being buffered or computed. Specifically, when the tasks are buffered, it means that the sum of the number of layers computed on the MD and the server is equal to 0. However, when the tasks are processed, the sum is the amount of layers of the DNN model L_u on the u th MD. The constraints (17) and (18) illustrate that the time taken by the u th MD and the server to process a unit of data is not less than the minimum value, respectively. The constraint (19) bounds the maximum data transmission rate from the u th MD to the server. The constraint (20) bounds the maximum data downloading rate from the server to the u th MD.

As discussed above, the decision taken by the u th MD is represented by $a_u(t) = [\alpha_{u,L}(t), \alpha_{u,M}(t)]$. The current state $s_u(t)$ can be obtained by the u th MD through observing the system, which is composed of the arrived tasks $D_u(t)$, the channel power gain between the u th MD and the server $h_u(t)$, and the tasks stored in buffer $Q_u(t)$. Thus, the current state $s_u(t)$ could be represented as $s_u(t) = [D_u(t), h_u(t), Q_u(t)] \in S$, and S is the state collection. In time slot t , the u th MD computes the immediate reward $R_u(t)$ after performing the action $a_u(t)$ in the state $s_u(t)$. Since we aim to finish the DNN tasks fast and ensure the minimum energy consumption, we define the reward function $R_u(t)$ as the total cost $C_u(t)$ consumed by the u th MD, i.e., $R_u(t) = C_u(t)$. Therefore, according to the action collection A , state collection S , and the immediate reward $R_u(t)$, we can model the optimization problem as an MDP problem.

Our goal is to enable each MD to learn the optimal strategy π^* , i.e., $a = \pi^*(s)$. Concretely, we have

$a = (a_1, a_2, a_3, \dots, a_n)$ and $s = (s_1, s_2, s_3, \dots, s_n)$. Since the arrival of tasks by the u th MD $D_u(t)$ is random in our system model, we don't know the probability distribution function (PDF). Therefore, we propose to use DRL to solve the MDP problem without explicitly specifying the transition probabilities.

DRL-based algorithm design

If we use the traditional reinforcement learning algorithm Q-learning to tackle the MDP problem described in the preceding section, it will produce a huge number of states, and the problem of dimensional disaster will be encountered in continuous tasks. To overcome this difficulty, we design a DRL-based DNN partitioning and task offloading (DPTO) algorithm. We use a policy-based Proximal Policy Optimization (PPO) algorithm with two neural networks, which can effectively address the above optimization problem **P1**. Figure 2 illustrates the framework of the DPTO algorithm.

The fundamental idea of our approach is to incorporate neural network technology to resolve the original MDP problem. The input to the neural network is the present state $s(t)$ of the agent, and then the corresponding action $a(t)$ is obtained. By taking the action $a(t)$, the next state $s(t+1)$ of the agent is updated and the reward value $R_u(t)$ is computed. According to the objective function including reward and action, the weight parameters in the neural network are updated by gradient rise, so as to obtain the action decision that makes the overall reward value smaller.

Our approach utilizes two distinct neural networks, namely actor and critic. The state makes the action decision through the actor network, in which the action is a probability value obtained after softmax, and then after sampling, the index value of the action will be obtained. However, the state obtained an expectation of the rewards of the agent through the critic network. According to the status, actions and rewards obtained at each time step, we update the parameters of the network.

Policy Gradient is used to update the parameters of the policy to minimize the cumulative reward. Specifically, the goal of the Policy Gradient is to compute the probability distribution of each action and to choose the action through this distribution in order to minimize the cumulative reward. We take the following gradient estimator:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_{\theta} \log \pi_{\theta}(a(t)|s(t)) \hat{R}(t) \right], \quad (21)$$

where π_{θ} represents a random policy parameterized by θ , $\hat{R}(t)$ denotes an estimation of the advantage function in time slot t , and the expectation $\hat{\mathbb{E}}_t$ represents the empirical average of limited batch of samples.

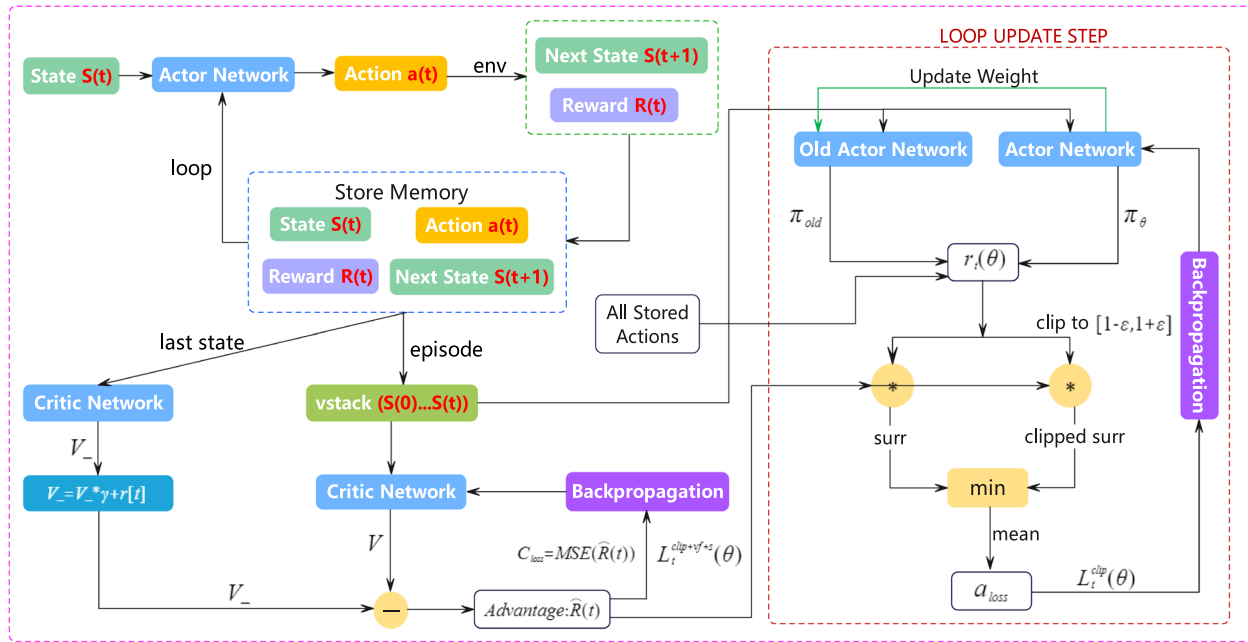


Fig. 2 Framework of DPTO algorithm

The learning ratio is a hyperparameter that governs the extent to which the parameters in the algorithm are updated, and it determines the step size of each parameter update. We introduce a measure to quantify the difference between the previous and updated policies, and define it as the probability ratio of actions under the two policies, as shown below:

$$r_t(\theta) = \frac{\pi_\theta(a(t)|s(t))}{\pi_{\theta_{old}}(a(t)|s(t))}. \quad (22)$$

Generating the policy distribution and selecting actions based on that distribution are the responsibilities of the actor network. During training, the actor network is updated by adjusting its parameters to minimize the expected reward. However, to prevent the policy from changing too drastically and causing instability, the update is limited within an acceptable ratio of ϵ , which is controlled by a hyperparameter called the clipping parameter. Based on PPO-Clip algorithm, the loss function of the actor network is denoted as follows:

$$L_t^{clip}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{R}(t), \text{clip}(\hat{R}(t)) \right) \right], \quad (23)$$

where $\text{clip} = \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)$. The clip function bounds the excessive update, which can prevent the bad policy of agent caused by the uncertainty of Monte Carlo sampling.

Input: Training episode number N ; training step number T ; decay factor ζ ; model update number $update_time$; the size of batch $batch_size$;

Output: The number of layers of DNN tasks executed locally $\alpha_L(t)$; the number of layers of DNN tasks executed on the MEC server $\alpha_M(t)$.

- 1: Initialize random seed, replay memory D , network parameters θ , system environment
- 2: **for** $episode = 1, \dots, N$ **do**
- 3: Observe the system environment
- 4: Get the initial state of the environment $s(0)$
- 5: Set reward = 0
- 6: **for** $step = 1, \dots, T$ **do**
- 7: Obtain action $a(t)$ from the actor network
- 8: Obtain the next state $s(t+1)$ with $a(t)$, compute reward value $R(t)$
- 9: Store $\langle s(t), a(t), R(t), s(t+1) \rangle$ into D
- 10: **if** $T \geq batch_size$ **then**
- 11: **for** $i = 1, \dots, update_time$ **do**
- 12: Sample random of transitions from D
- 13: Compute the decay reward $discount_reward = r + \zeta * discount_reward$
- 14: $r \leftarrow discount_reward$
- 15: Compute the advantage function using Eq. (24)
- 16: Compute the optimization goals using Eq. (26)
- 17: Update network parameters θ
- 18: **end for**
- 19: Save the updated network parameters $\pi_{old} \leftarrow \pi_\theta$
- 20: **end if**
- 21: **end for**
- 22: **end for**

Algorithm 1 The DNN partitioning and task offloading

(DPTO) algorithm We update the critic network using data from the experience pool after updating the actor network. To achieve this, we compute the advantage function using the experience data, which represents the difference between the anticipated reward of taking an action and the anticipated reward of following the current policy. Specifically, the advantage function can be mathematically defined using the following formula:

$$\hat{R}(t) = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \quad (24)$$

where,

$$\delta_t = R(t) + \gamma V(s(t+1)) - V(s(t)). \quad (25)$$

And we set $\lambda = 1$. The value of $V(s(t+1))$ and $V(s(t))$ can be obtained by utilizing the critic network. Given the neural network structure we adopt, which shares parameters between the policy and the value function, a loss function is required that incorporates the policy agent and the value function error. Specifically, we define the loss function as follows:

$$\mathbf{L}_t^{clip+vf+s}(\theta) = \mathbb{E}_t \left[\mathbf{L}_t^{clip}(\theta) - c_1 \mathbf{L}_t^{vf}(\theta) + c_2 Z \right], \quad (26)$$

where c_1 and c_2 are coefficients. Z represents the entropy reward of the possibility of global exploration of new policies by the critic network and $Z = S[\pi_\theta](s(t))$. $\mathbf{L}_t^{vf}(\theta)$ is the square difference loss. This goal can be further enhanced by increasing entropy reward to ensure full exploration. This goal is approximately minimized in each iteration.

Algorithm 1 provides a detailed description of the DPTO algorithm.

Performance evaluation

This section presents the results of extensive simulation experiments carried out to validate the effectiveness of our proposed DPTO algorithm. The open source machine learning framework Pytorch in Python was used for constructing and training the neural networks.

Experimental setup

Metrics

The evaluation of our proposed method is based on three defined metrics: processing delay, energy consumption, and system cost, which are specified in Eqs. (10), (13), and (14), respectively. To further evaluate the performance of our proposed method, we compare it with existing non-DRL and DRL algorithms.

Parameter setting

We adopt some of the parameter settings used in [14] for our simulation experiments. Optimizing the

hyperparameters of DRL network training is an ongoing process that requires continuous adjustments to achieve the best convergence and performance. Specifically, we assume that the MEC system architecture consists of one MEC server and five MDs, where we use the Orange Pi Win Plus as the MD, and the computing power and offloading power of each MD are 4.05 W and 4 W, respectively. We consider the MEC server to have a transmission power of 600 W and set the available channel bandwidth between the u th MD and the MEC server to 10 KHz. The power spectral density of noise is -30 dbm/KHz. The u th MD and the MEC server require different amounts of time to process one KB of data, with processing times ranging from 0.0001 to 0.001 seconds and 0.00001 to 0.00005 seconds, respectively.

According to [19], we suppose that the channel power gain between the u th MD and the server, which is denoted as $h_u(t)$, follows the Markov property. Specifically, we have $P(m_u/10000 \leq h_u(t+1) \leq m_u | h_u(t) = m_u) = 0.9$ and $P(0 \leq h_u(t+1) \leq m_u/10000 | h_u(t) = m_u) = 0.1$, where $m_u = 1.2 \times 10^6$. The computational tasks $D_u(t)$ are randomly generated, with sizes randomly selected from the range 1 MB to 5 MB. Considering that it is possible for a subtask producing output data that is less or more than its original input data, the value of the parameter $f_{u,l}$ is varied from 0.1 to 2 [35]. For the three weights of the total cost $C_u(t)$ of the u th MD, we set $\mu = 20$, $\nu = 5$, $\omega = 1$. We train four DNN models with different computation difficulty over the cifar-10 dataset [40], including VGG16, VGG13, ALEXNET, and LENET, and the difficulty decreased from VGG16 to LENET in turn. Table 2 provides a detailed overview of the parameter settings used in our system.

Experimental results

Comparison experiments with non-DRL algorithms

To evaluate the performance of our proposed DPTO algorithm, we compare it with the following three traditional non-DRL algorithms, where the adopted DNN is the VGG16 model.

(1) Local execution: The computation of all layers of DNN tasks is processed on local MDs.

(2) Offloading execution: The computation of all layers of DNN tasks is offloaded to the MEC server for processing.

(3) Random: The DNN tasks are randomly layered and offloaded to the server for computation.

The average processing delay and average energy consumption of the four algorithms under different bandwidths are illustrated in Fig. 3(a) and (b), respectively. From the figure, we can see that the delay and energy consumed by DNN tasks computed on local MDs do not change with the change in bandwidth. Since local

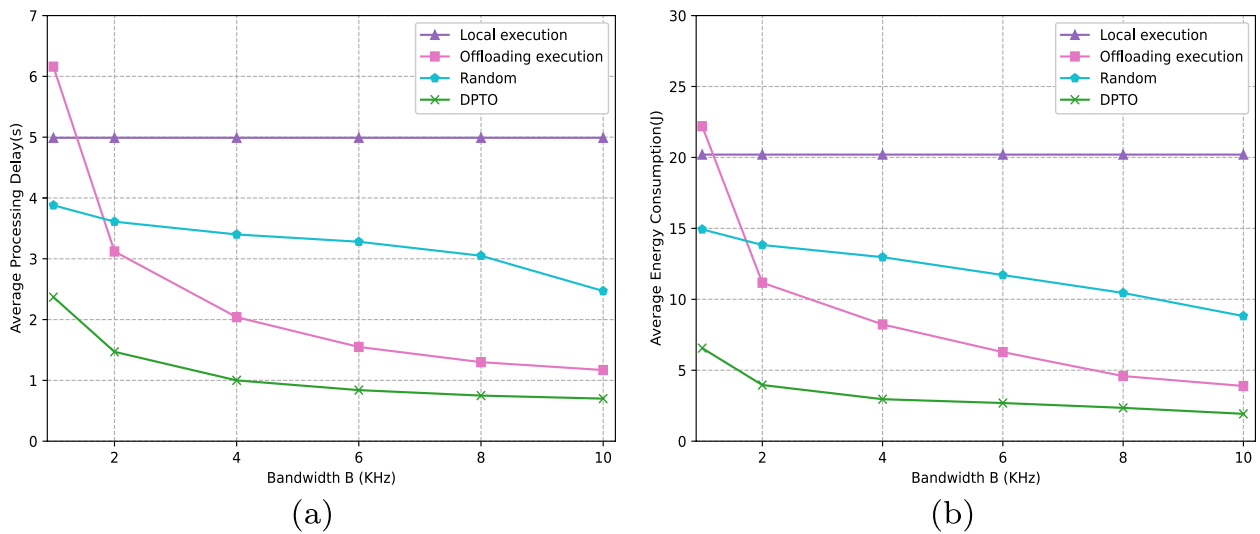


Fig. 3 Comparisons of different algorithms under different bandwidths. (a) Average processing delay. (b) Average energy consumption

execution does not need to offload data to the server, it has nothing to do with bandwidth. However, in the other three methods, the average processing delay and average energy consumption decrease as the bandwidth increases. As the bandwidth between the MD and the server increases, the data uploading and downloading rates will also increase, reducing average processing delay and energy consumption. When there are sufficient bandwidth resources in the dynamic network environment, we tend to offload subtasks to the server for processing. On the contrary, when bandwidth resources are scarce, we tend to compute subtasks on local MDs. These two figures also show that the DPTO algorithm is

superior to the other three non-DRL algorithms. This is because the DPTO algorithm utilizes deep neural networks as strategy functions to improve strategies through backpropagation and gradient optimization to better adapt to complex environments.

Figure 4 shows the comparisons of four different algorithms under different DNN types, and verify the scalability of our DPTO algorithm to various DNN types. Figure 4(a) indicates how the average processing delay varies with various types of DNNs, and Fig. 4(b) indicates how the average energy consumption varies with various types of DNNs. By analyzing these two figures, we can notice that the trend is that the average processing

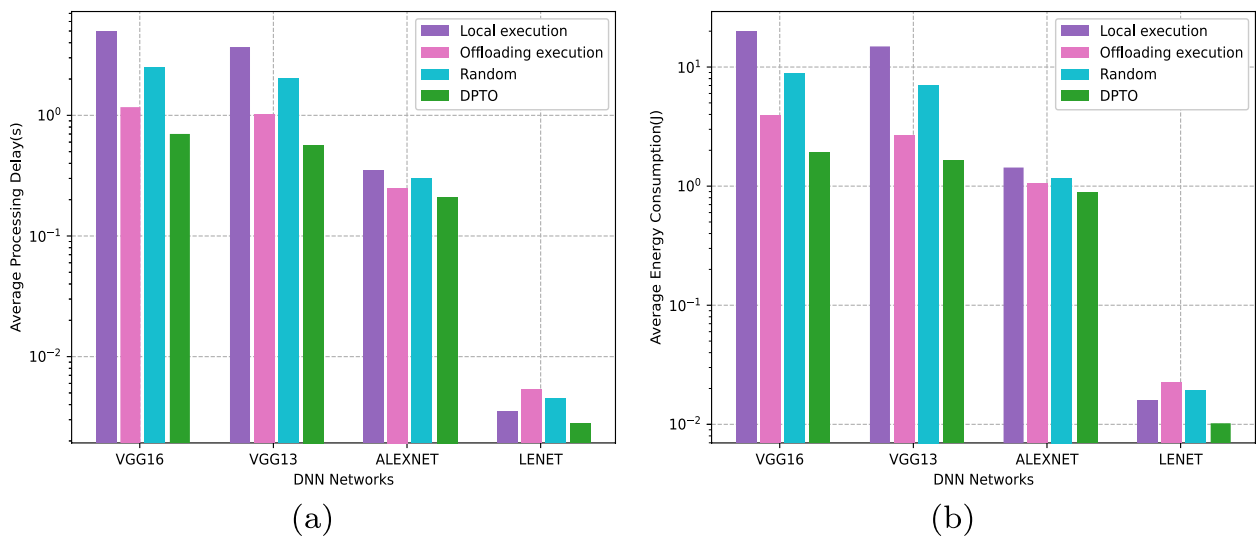


Fig. 4 Comparisons of different algorithms under different DNN types. (a) Average processing delay. (b) Average energy consumption

Table 2 Simulation parameter setting

Parameter	Value
n	5
p_u^{exe}	4.05 W
p_u^{up}	4 W
P_M	600 W
D_u	1MB ~ 5MB
B_u	10 KHz
N_0	-30 dbm/KHz
h_u	1.2×10^6
$\xi_{u,loc}$	[0.0001, 0.001]
ξ_{mec}	[0.00001, 0.00005]
$f_{u,l}$	0.1 ~ 2

delay and energy consumption of all four algorithms decrease as the computation difficulty of the DNN decreases. When dealing with DNN tasks that have relatively high computation difficulty (i.e., VGG16, VGG13, and ALEXNET), local execution can be both time-consuming and energy-consuming because the computation and energy resources available to local MDs are limited. Conversely, when dealing with DNN tasks that have relatively low computation difficulty (i.e., LENET), off-loading execution and random execution become slow and energy-consuming due to high uploading delay and energy consumption. And we can also see that the DPTO algorithm is always outperforming other three non-DRL algorithms with respect to the average processing delay and energy consumption, regardless of the type of DNNs. This is because the DPTO algorithm provides flexible policy optimization methods, adopts adaptive adjustment of hyperparameters, and considers real-time conditions and optimization objectives.

Comparison experiments with DRL algorithms

The following experiments will compare our DPTO algorithm with two commonly utilized DRL algorithms, which are used to resolve dynamic optimization problems that have a discrete action space. These two algorithms are widely used and are listed below.

(1) Deep Q-Network (DQN): DQN is a DRL algorithm based on value rather than policy. Its primary idea is to use neural network techniques to estimate the Q-value function, which helps to solve reinforcement learning problems that have a high-dimensional state space. In this algorithm, the neural network takes the environmental state as input and produces the Q-value for every feasible action as output. The algorithm follows the ϵ -greedy strategy to choose an action and updates the neural network parameters to minimize the objective function at

each time step. In the updating process, it uses the experience replay technology to alleviate the data correlation problem, and at the same time uses the target network to reduce the fluctuation of the objective function.

(2) Double Deep Q-Network (DDQN): DDQN aims to overcome the overestimation problem of Q-value in the DQN algorithm. In this algorithm, the Q-network parameters used when selecting the action and fitting the target are not the same set of parameters, but parameters at different times, which can decouple the selecting action from the evaluating action. It trains two Q networks and selects the smaller Q-value to compute TD-error at the same time, which could reduce the overestimation error. In addition, by using the output of the evaluation network to determine the optimal action of the target network, the DDQN algorithm can more effectively mitigate the overestimation problem.

We train our proposed method and two other DRL algorithms for 500 iterations, and compare their convergence rate and performance according to experimental results. To achieve stable training and efficient learning, we set both the DQN and DDQN algorithms to have an experience pool size of 10000 and a batch size of 200. The reward values during the initial 500 epochs are presented in Fig. 5. And the experimental results demonstrate that the DPTO and DDQN algorithms achieve gradual convergence within the first 34 and 110 epochs, respectively. In contrast, the DQN algorithm does not converge even after 500 iterations. Because the DQN algorithm uses maximization operations to select actions, which can lead to the problem of overestimating the value function. As shown in Fig. 5, out of the three algorithms, the DPTO algorithm achieves the fastest convergence speed and the lowest reward value. On the one hand, this is because the DPTO algorithm limits the amplitude of policy updates in each update, which can keep policy updates within a controllable range. On the other hand, the DPTO algorithm optimizes the policy directly and uses multiple sampling trajectories for policy updates.

Figure 6 gives the processing costs of the DPTO algorithm and the other two DRL algorithms under different bandwidths. We can notice that as the bandwidth between the MD and the server increases, the processing costs of the three algorithms all decrease. The DQN algorithm is the worst and most unstable. This is due to the fact that there are some differences between the target network and the action network in the DQN algorithm, which leads to unstable training and non-convergence, and also the DQN algorithm needs to represent the state as a fixed-length vector, which limits the expressive ability of the state space and leads to poor performance in some tasks. Our DPTO algorithm has the best performance under different bandwidths.

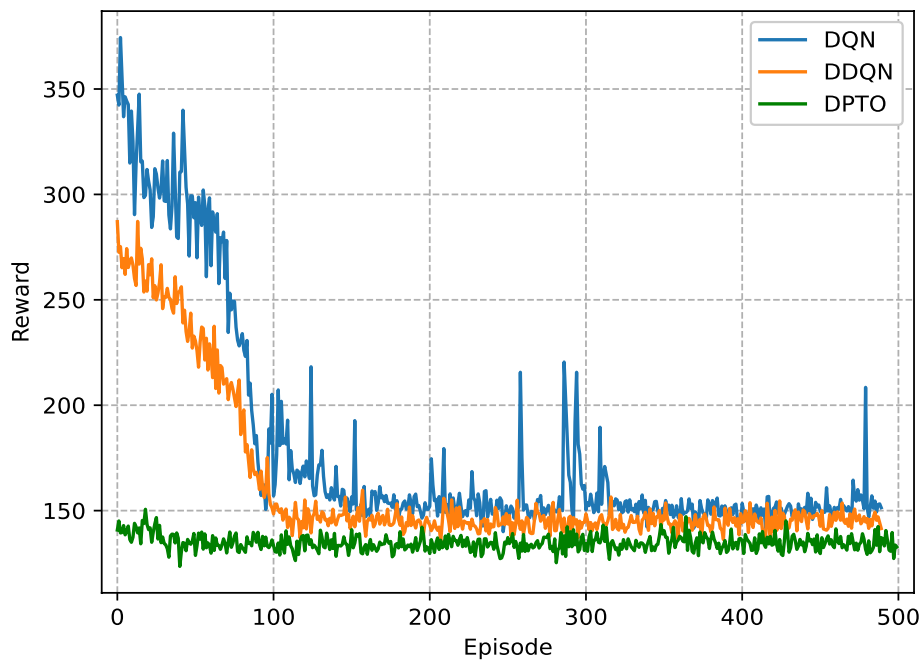


Fig. 5 Comparisons of rewards for different DRL algorithms during training

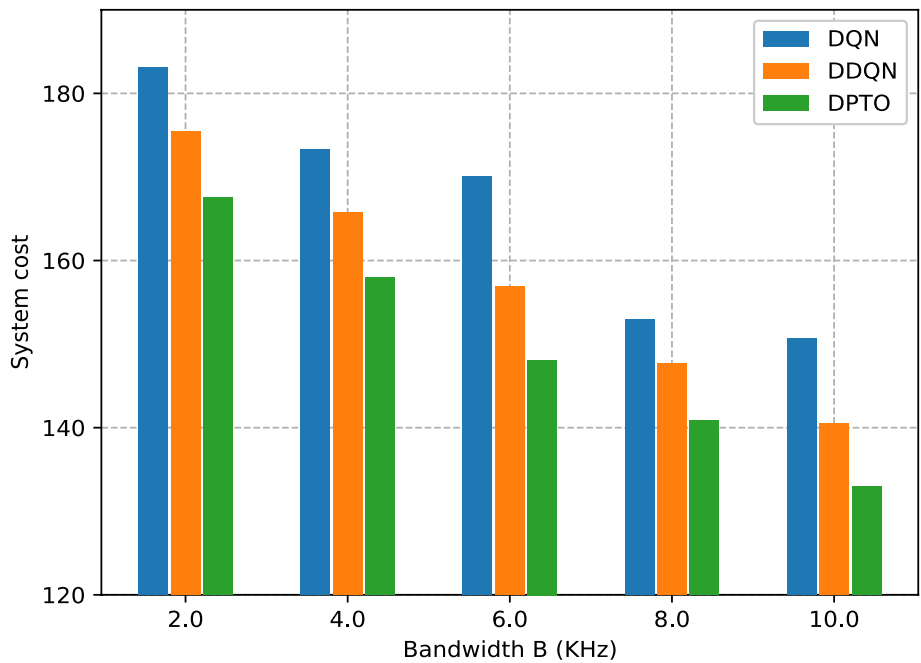


Fig. 6 System cost under different bandwidths

Because the DQN and DDQN algorithms use the greedy strategy based on the Q-value to optimize the strategy. However, the DPTO algorithm optimizes the strategy by constraining the maximum and minimum

values of the objective function, which can better ensure the stability and convergence of the strategy.

Figure 7 indicates the comparisons of three different DRL algorithms under different DNN types, where

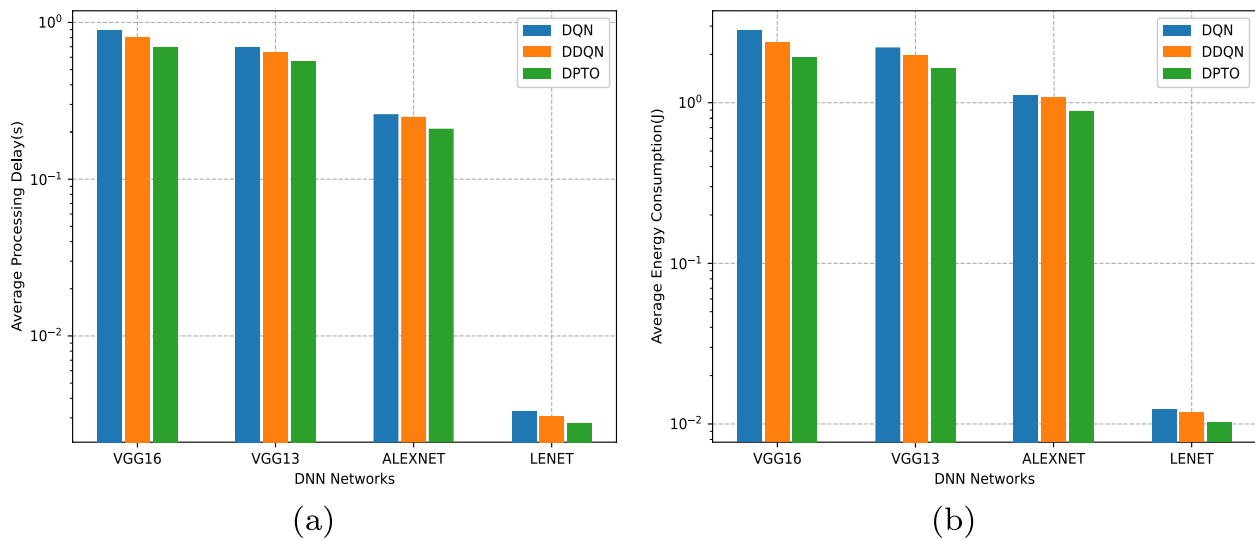


Fig. 7 Comparisons of different DRL algorithms under different DNN types. (a) Average processing delay. (b) Average energy consumption

Fig. 7(a) displays how the average processing delay varies with various DNN types, and Fig. 7(b) displays how the average energy consumption varies with various DNN types. Obviously, we can see that the three DRL algorithms have lower average processing delay and energy consumption when the DNN computation difficulty is lower. This is because the higher the computation difficulty of DNN tasks, the higher the computational delay and energy consumption. Furthermore, our DPTO algorithm consistently outperforms the DQN and DDQN algorithms in both processing delay and energy consumption, regardless of the DNN types. Because the DPTO algorithm is based on PPO, and PPO uses online data directly for training, it can make efficient use of sampled data and has better stability.

Conclusion

This paper investigates the joint optimization of energy and delay for DNN partitioning and task offloading in a MEC system consisting of a MEC server and multiple MDs with buffers. We partition the DNN tasks into sub-tasks by layer and offload all or part of them to the server for processing. Then, we formulate the processing delay and energy consumption as a joint optimization problem and further model it as an MDP problem. To tackle this problem, we design a DRL-based approach, which can help MDs to choose the best offloading policy. Finally, through a large number of experiments, we find that our DPTO algorithm achieves superior performance in minimizing both processing delay and energy consumption compared to the existing non-DRL and DRL algorithms, and can be extended to different DNN types.

In the future, we will investigate DNN partitioning and task offloading in a scenario with multiple MDs and multiple servers. We will also explore other optimization techniques to further improve the performance of DNN partitioning and task offloading in the context of MEC systems. Furthermore, considering the importance of privacy issues during task offloading, we will delve into the content related to privacy issues in DNN partitioning and task offloading.

Abbreviations

AI	Artificial intelligence
DNNs	Deep neural networks
MDs	Mobile devices
MEC	Mobile edge computing
MDP	Markov decision process
DPTO	DNN partitioning and task offloading
DRL	Deep reinforcement learning

Acknowledgements

We would like to express our sincere thanks to all reviewers. The reviewers' comments and suggestions have played a positive role in improving the quality of our paper.

Authors' contributions

Jianbing Zhang was mainly responsible for the conception of this manuscript and the design of the system model. Shufang Ma contributed to the design and implementation of the system modelling methods and algorithms. In addition, she also helped to revise the manuscript. Zexiao Yan was in charge of the experimental data collection and analysis, as well as the execution of the experiments. Jiwei Huang put forward many valuable suggestions on the motivation and significance of the manuscript and proofread the final version. The final manuscript was reviewed and approved by all authors.

Authors' information

Jianbing Zhang received the Ph.D. degree from the Institute of Remote Sensing and Digital Earth at Chinese Academy of Sciences in 2006. He is now an assistant professor in the Department of Computer Science and Technology at the China University of Petroleum, Beijing, China. His current research interests include web services and GIS services.

Shufang Ma is currently working toward the M.Eng. Degree in computer technology at the China University of Petroleum, Beijing, China. Her current research interests include edge intelligence and deep reinforcement learning. Zexiao Yan is currently working toward the M.Eng. Degree in computer technology at the China University of Petroleum, Beijing, China. His current research focuses on deep learning and change detection.

Jiwei Huang received the B.Eng. and Ph.D. degrees in computer science and technology from Tsinghua University, Beijing, China, in 2009 and 2014, respectively. He is currently a Professor and the Dean with the Department of Computer Science and Technology, China University of Petroleum, Beijing, China. His research interests include services computing, Internet of Things, and edge computing.

Funding

This work is supported by National Natural Science Foundation of China (No. 61972414) and Beijing Nova Program (No. Z201100006820082).

Availability of data and materials

Data are available on the website: Cifar-10: <https://www.cs.toronto.edu/~kriz/cifar.html>.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare no competing interests.

Received: 6 May 2023 Accepted: 25 July 2023

Published online: 03 August 2023

References

- Chen J, Ran X (2019) Deep learning with edge computing: A review. *Proc IEEE* 107(8):1655–1674
- Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp 1–9. <https://doi.org/10.1109/CVPR.2015.7298594>
- Wang D, Nyberg E (2015) A long short-term memory model for answer sentence selection in question answering. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers). Association for Computational Linguistics, Beijing, China. p 707–712. <https://doi.org/10.3115/v1/P15-2116>
- Chen Z, Hu J, Chen X, Hu J, Zheng X, Min G (2020) Computation offloading and task scheduling for dnn-based applications in cloud-edge computing. *IEEE Access* 8:115537–115547
- Mao Y, You C, Zhang J, Huang K, Letaief KB (2017) A survey on mobile edge computing: The communication perspective. *IEEE Commun Surv Tutor* 19(4):2322–2358
- Mach P, Becvar Z (2017) Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun Surv Tutor* 19(3):1628–1656
- Xiao Z, Dai X, Jiang H, Wang D, Chen H, Yang L, Zeng F (2020) Vehicular task offloading via heat-aware MEC cooperation using game-theoretic method. *IEEE Internet Things J* 7(3):2038–2052
- Lin L, Liao X, Jin H, Li P (2019) Computation offloading toward edge computing. *Proc IEEE* 107(8):1584–1607
- Li E, Zeng L, Zhou Z, Chen X (2020) Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Trans Vis Comput Graph* 19(1):447–457
- Xu D, Li T, Li Y, Su X, Tarkoma S, Jiang T, Crowcroft J, Hui P (2021) Edge intelligence: Empowering intelligence to the edge of network. *Proc IEEE* 109(11):1778–1837
- Tang X, Chen X, Zeng L, Yu S, Chen L (2021) Joint multiuser DNN partitioning and computational resource allocation for collaborative edge intelligence. *IEEE Internet Things J* 8(12):9511–9522
- Dong F, Wang H, Shen D, Huang Z, He Q, Zhang J, Wen L, Zhang T (2022) Multi-exit DNN inference acceleration based on multi-dimensional optimization for edge intelligence. *IEEE Trans Mob Comput* 1. <https://doi.org/10.1109/TMC.2022.3172402>
- Dong C, Hu S, Chen X, Wen W (2021) Joint optimization with DNN partitioning and resource allocation in mobile edge computing. *IEEE Trans Netw Serv Manag* 18(4):3973–3986
- Gao M, Shen R, Shi L, Qi W, Li J, Li Y (2023) Task partitioning and offloading in DNN-task enabled mobile edge computing networks. *IEEE Trans Mob Comput* 22(4):2435–2445
- Li W (2020) Geoai: Where machine learning and big data converge in gisience. *J Spat Inf Sci* 20:71–77
- Li W, Batty M, Goodchild MF (2020) Real-time GIS for smart cities. *J Geog Inf Sci* 34(2):311–324
- Zhou Z, Chen X, Li E, Zeng L, Luo K, Zhang J (2019) Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc IEEE* 107(8):1738–1762
- Kang Y, Hauswald J, Gao C, Rovinski A, Mudge T, Mars J, Tang L (2017) Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *SIGARCH Comput Archit News* 45(1):615–629
- Zhang G, Ni S, Zhao P (2022) Learning-based joint optimization of energy delay and privacy in multiple-user edge-cloud collaboration MEC systems. *IEEE Internet Things J* 9(2):1491–1502
- He W, Guo S, Guo S, Qiu X, Qi F (2020) Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT. *IEEE Internet Things J* 7(10):9241–9254
- Li K, Zhao J, Hu J et al (2022) Dynamic energy efficient task offloading and resource allocation for noma-enabled IoT in smart buildings and environment. *Build Environ*. <https://doi.org/10.1016/j.buildenv.2022.109513>
- Chen Y, Zhao J, Wu Y et al (2022) Qoe-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach. *IEEE Trans Mob Comput*. <https://doi.org/10.1109/TMC.2022.3223119>
- Huang J, Wan J, Lv B et al (2023) Joint computation offloading and resource allocation for edge-cloud collaboration in internet of vehicles via deep reinforcement learning. *IEEE Syst J*. <https://doi.org/10.1109/JSYST.2023.3249217>
- Li J, Liang W, Li Y, Xu Z, Jia X, Guo S (2023) Throughput maximization of delay-aware DNN inference in edge computing by exploring DNN model partitioning and inference parallelism. *IEEE Trans Mob Comput* 22(5):3017–3030
- Liu G, Dai F, Huang B, Qiang Z, Wang S, Li L (2022) A collaborative computation and dependency-aware task offloading method for vehicular edge computing: a reinforcement learning approach. *J Cloud Comput* 11
- Zhang J, Ma B, Huang J (2020) Deploying GIS services into the edge: A study from performance evaluation and optimization viewpoint. *Secur Commun Netw* 2020:1–13
- Chen Y, Gu W, Xu J, et al (2022) Dynamic task offloading for digital twin-empowered mobile edge computing via deep reinforcement learning. *China Commun*. <https://doi.org/10.23919/JCC.ea.2022-0372.202302>
- Huang J, Gao H, Wan S, et al (2023) Aoi-aware energy control and computation offloading for industrial IoT. *Futur Gener Comput Syst* 139:29–37
- Li S, Zhang N, Jiang R, Zhou Z, Zheng F, Yang G (2022) Joint task offloading and resource allocation in mobile edge computing with energy harvesting. *J Cloud Comput Adv Syst Appl* 11(1):1–14
- Zhou Y, Ge H, Ma B et al (2022) Collaborative task offloading and resource allocation with hybrid energy supply for UAV-assisted multi-clouds. *J Cloud Comput* 11. <https://doi.org/10.1186/s13677-022-00317-2>
- Chen Y, Hu J, Zhao J, Min G (2023) Qos-aware computation offloading in LEO satellite edge computing for IoT: A game-theoretical approach. *Chin J Electron*. <https://doi.org/10.23919/cje.2022.00.412>
- Zhang G, Zhang W, Cao Y, Li D, Wang L (2018) Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices. *IEEE Trans Ind Inform* 14(10):4642–4655

33. Xu J, Chen L, Zhou P (2018) Joint service caching and task offloading for mobile edge computing in dense networks. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. p 207–215. <https://doi.org/10.1109/INFOCOM.2018.8485977>
34. Chen L, Zhou S, Xu J (2018) Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. *IEEE/ACM Trans Networking* 26(4):1619–1632
35. Xu Z, Zhao L, Liang W, Rana OF, Zhou P, Xia Q, Xu W, Wu G (2021) Energy-aware inference offloading for DNN-driven applications in mobile edge clouds. *IEEE Trans Parallel Distrib Syst* 32(4):799–814
36. Chen J, Chen S, Wang Q, Cao B, Feng G, Hu J (2019) iraf: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks. *IEEE Internet Things J* 6(4):7011–7024
37. Jeong HJ, Lee HJ, Shin CH, Moon SM (2018) lonn: Incremental offloading of neural network computations from mobile devices to edge servers. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'18)*. Association for Computing Machinery, New York. p 401–411. <https://doi.org/10.1145/3267809.3267828>
38. Yang Q, Luo X, Li P, Miyazaki T, Wang X (2019) Computation offloading for fast CNN inference in edge computing. In *Proceedings of the Conference on Research in Adaptive and Convergent Systems (RACS'19)*. Association for Computing Machinery, New York. p 101–106. <https://doi.org/10.1145/3338840.3355669>
39. Shannon CE (2001) A mathematical theory of communication. *ACM SIGMOBILE Mob Comput Commun Rev* 5(1):3–55
40. Krizhevsky A, Hinton G, Sutskever I (2009) Learning multiple layers of features from tiny images. Tech. Rep. TR2009-08, Computer Science Department, University of Toronto

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- ▶ Convenient online submission
- ▶ Rigorous peer review
- ▶ Open access: articles freely available online
- ▶ High visibility within the field
- ▶ Retaining the copyright to your article

Submit your next manuscript at ► [springeropen.com](https://www.springeropen.com)
