A Dual Attention-Based Task Offloading Approach in Computing Power Networks for Object Detection

Kang Huang[1], Chao Qiu[1], Hong Zhu[3], Lisha Gao[3], Qizhe Zhang[2], Guozheng Peng[2], Nan Xiang[3], and Xiaofei Wang[1(✉)]

[1] College of Intelligence and Computing, Tianjin University, Tianjin, China
xiaofeiwang@tju.edu.cn
[2] China Electric Power Research Institute, Beijing, China
[3] State Grid Nanjing Power Supply Company, Nanjing, China

Abstract. Intelligent object detection has enabled automatic defect detection at the edges, facilitating smart power transmission inspection and target recognition. However, edge devices often suffer from limited computational resources, resulting in slow model recognition and high energy consumption, making it challenging to meet daily inspection requirements. Computing Power Networks (CPNs) provide a secure and efficient distributed computing solution, enabling the effective offloading of object detection tasks to resource pools within CPNs. Despite numerous studies focusing on optimizing task offloading, some issues persist, including coarse granularity of computing tasks, dispersed computing resources, and weak decision adaptability in complex environments. To address these challenges, we propose a dual attention-based deep reinforcement learning (Dat-DRL) approach, utilizing a custom sequence-to-sequence (seq2seq) neural network to learn effective task offloading strategies. Monitoring tasks at the edge of the power transmission line are modeled as a directed hypergraph, and the dual attention mechanism captures inter-task dependencies and diverse service requirements more effectively. To evaluate the proposed algorithm, we construct a simulation environment to model task offloading scenarios, and extensive experiments demonstrate the efficacy of Dat-DRL in reducing latency and energy consumption across different environments.

Keywords: Object detection · Computing power networks · Task offloading · Hypergraph · Dual attention-based

1 Introduction

With the continuous development of edge intelligence [22], many monitoring and intelligent recognition tasks can be achieved through the computing power services provided by edge cloud devices. Defect perception at the edge requires intelligent monitoring. However, due to the limited computational and storage capabilities of edge devices and the heterogeneity and dispersion of edge cloud nodes,

V. C. M. Leung et al. (Eds.): Qshine 2023, LNICST 574, pp. 244–263, 2024.
https://doi.org/10.1007/978-3-031-65123-6_18

the distribution of monitoring tasks is uneven. As a result, only lightweight models can be executed, which fails to meet the efficient business requirements for defect perception. Computing Power Networks (CPNs) are a novel architecture that deeply integrates computing and networking [14], establishing a new information infrastructure with computation as its core and networking as its foundation. CPNs can dynamically allocate and schedule computing resources, storage resources, and network resources among cloud, network, and edge based on task requirements, which can provide assorted computing solutions [14] and better adapt to the heterogeneous and fragmented problem of defect perception in power transmission lines.

Energy consumption in CPNs has emerged as a core issue. Speculatively, data centers are predicted to become the world's largest energy consumers, with the ratio rising from 3% in 2017 to 4.5% in 2025. [10]. The task offloading problem in CPNs is closely related to energy consumption, and one of the significant contributors to increased energy consumption is the coarse-grained analysis of tasks and resources. Moreover, this coarse-grained analysis also leads to an increased risk of damage to edge devices and economic losses, which hinders the smooth operation of monitoring services for power transmission lines.

In recent years, several studies have been conducted to tackle the task offloading problem, and researchers have explored various techniques to optimize energy consumption. Zhai et al. [24] devised an energy-aware dynamic offloading scheme that effectively harnesses available battery power to execute a greater number of applications within the constraints of application dependencies. Zhao et al. [25] introduced an iterative algorithm aimed at optimizing the energy consumption of mobile edge computing while considering latency constraints. This algorithm has demonstrated the potential to achieve energy savings of 20% to 40%. Chen et al. [3] treated tasks as indivisible entities and formulated task offloading as a stochastic optimization problem, aiming to minimize task offloading energy consumption while ensuring the average queue length under uncertain infinite channel states. Furthermore, many studies have embraced the utilization of deep reinforcement learning(DRL) techniques in addressing the prevalent energy challenges [1,2,13].

Despite the efforts made in the above-mentioned studies to resolve energy consumption issues through task offloading using various algorithms, several critical challenges have been overlooked, encompassing:

- Task Aspect: There are various monitoring tasks in power transmission lines, and it is inappropriate to indiscriminately consider all tasks at the same granularity. Some workflow tasks in CPNs can be decomposed and reused. Many studies have not taken into account the existence of task dependencies, which can potentially lead to suboptimal optimization results.
- Resource Aspect: Decentralized resource management. As the number of resources in CPNs grows, identifying and efficiently scheduling appropriate resources from the vast pool becomes exceedingly difficult, leading to conflicts and uncertainty in task offloading.

– Task-Resource Adaptation: Weak decision generalization and adaptability. Traditional methods for addressing task offloading heavily rely on expert knowledge and precise mathematical models, limiting their adaptability to unexpected disturbances or unknown scenarios.

In response to the aforementioned problems and challenges, we first model monitoring tasks from different edge devices (ED) in power transmission lines as a directed hypergraph. A hypergraph is a data structure capable of representing multiple associative relationships. Unlike a regular graph, its edges can connect to multiple vertices. We propose a Dat-DRL algorithm for achieving task offloading in CPNs, to effectively handle task offloading. The dual-attention mechanism allows the neural network to filter out irrelevant information and focus on crucial features, thus enhancing the quality of offloading strategies and accelerating model convergence. The primary contributions of this paper are as follows:

1. We present an attention-based task offloading algorithm that addresses the complex problem of offloading tasks with dependencies in CPNs. To enhance energy efficiency, dependent tasks are offloaded to execute either locally or on the fine-grained resource pool.
2. In order to assign tasks from the directed hypergraph to appropriate CPN nodes in the resource pool, we employ a Dat-DRL algorithm, which rapidly adapts to tasks, allowing the model to disregard irrelevant noise and accelerate convergence by focusing on critical information through the dual-attention mechanism.
3. Extensive simulation and control experiments are conducted to evaluate the applicability of Dat-DRL in fine-grained task offloading. The results demonstrate that Dat-DRL outperforms various baselines, significantly reducing the number of training steps by over 53.40% compared to DRL models lacking the dual-attention mechanism. Regarding model performance, Dat-DRL reduces the overall energy consumption by over 28.24%.

2 Related Work

With the rise of cloud computing and edge computing, task offloading has been a subject of significant research. Many research works on task offloading in mobile edge computing can provide insights into the task offloading problem in CPNs. Several studies have employed heuristic algorithms. For instance, Fen et al. [23] represented the task offloading problem as a two-dimensional knapsack loading problem and introduced a suboptimal heuristic algorithm, which, compared to the baseline, enhances cost efficiency. Ming et al. [11] modeled the tasks within task unloading as a directed acyclic graph (DAG) and formulated the optimization problem as minimizing the weighted critical path of the DAG. They proposed a heuristic solution and validated its effectiveness through experiments. Lei et al. [8] tackled dynamic task optimization by employing an approximate dynamic programming approach to jointly optimize computation

offloading and multi-user scheduling. Their objective was to minimize long-term average weighted delay and power consumption under random traffic arrivals.

In addition to using heuristic algorithms, many works have employed algorithms based on DRL. DRL algorithms [2,7,12] are utilized in interactive learning with the environment and offer adaptability to complex and dynamic scenarios. Among them, policy optimization-based algorithms achieve effective optimization by learning both policy and value functions. For instance, trust region policy optimization (TRPO) [15] and proximal policy optimization (PPO) [17] ensure stability and reliability in policy updates. Various neural network models are used in DRL, including recurrent neural network models such as the classic long short-term memory network (LSTM) [5], which effectively addresses modeling and prediction of long sequence data, and the encoder-decoder structured Seq2Seq network model.

Due to the adaptability of DRL algorithms to task sequence offloading optimization problems in task offloading scenarios, many research works have applied these algorithms. For example, Wang et al. [20] proposed a deep reinforcement learning-based offloading framework utilizing a specially designed Seq2Seq neural network for unique representation of offloading strategies to adapt to dynamic changes and achieve optimal performance under different transmission rates and task quantities. Chai et al. [1] have proposed an algorithm for optimizing proximity-based strategies using single-layer attention to address the challenge of multi-task joint computation offloading. Di et al. [4] efficiently utilize idle resources from a dynamic resource pool to enhance the system's utility through attention-based proximal policy optimization. Unlike the DRL algorithms mentioned earlier, we employ a dual attention mechanism, which better captures the interrelationships between tasks and their resource requirements.

3 System Model

The proposed algorithm requires learning in a simulated environment. To achieve this, we establish an environment module that simulates the task offloading scenario, including the ED local system and the resource pool system.

3.1 Object Detection Task Model in ED System

As shown in Fig. 1, each ED system consists of one or multiple object detection tasks that can be represented as a directed hypergraph: $\mathcal{H} = (\mathcal{T}, \mathcal{E})$, where $\mathcal{T} = \{t_1, t_2, \ldots, t_i, \ldots, t_n\}$ represents the set of tasks, and $\mathcal{E} = \{e_1, e_2, \ldots, e_j, \ldots, e_m\}$ is the set of directed hypergraphs representing the dependencies between task, $n = |\mathcal{T}|$ denotes the number of tasks, $m = |\mathcal{E}|$ denotes the quantity of hyperedges. Specifically, tasks t_0 and t_e are defined as the start and end points of the task, respectively, we will use a hyperedge to connect all task nodes and t_e, indicating that the end task always lags behind all other tasks. The i-th task t_i can be represented by $\{C_i, inData_i, outData_i, type_i\}$, where $C_i, inData_i, outData_i$

respectively denote the computational requirements, input data size, and output data size of the task. Additionally, we define a set of task types $\mathcal{V} = \{1, 2, \ldots, v, \ldots, k\}$ to represent the types of tasks. $type_i \in \mathcal{V}$ represents the type of task. We assume that there are a total of k types of tasks. The j-th hyperedge $e_j = \{pred_j, eW_j, t_j, eType_j\}$ signifies a directed hyperedge directed towards the task t_j. $pred_j$ indicates the set of tasks that must be completed before initializing task t_j, while eW_j measures the data transfer time to task t_j by these preceding tasks. To convey additional relationships between tasks, $eType_j = \{0, 1\}$ signifies whether the nodes connected by the hyperedge can be offloaded to different CPN nodes.

The adjacency matrix H of a hypergraph can be represented as a matrix of dimensions $n \times m \times 2$. We represent the relationship between the i-th task and the j-th hyperedge with a vector h_ij of length 2:

$$h_{ij0} = \begin{cases} eW_j, & \text{if } t_i \in pred_j, \\ -1, & \text{if } t_i == t_j, \\ 0, & \text{else}, \end{cases} \tag{1}$$

and $h_{ij1} = eType_j$ is used to represent the type of the hyperedge.

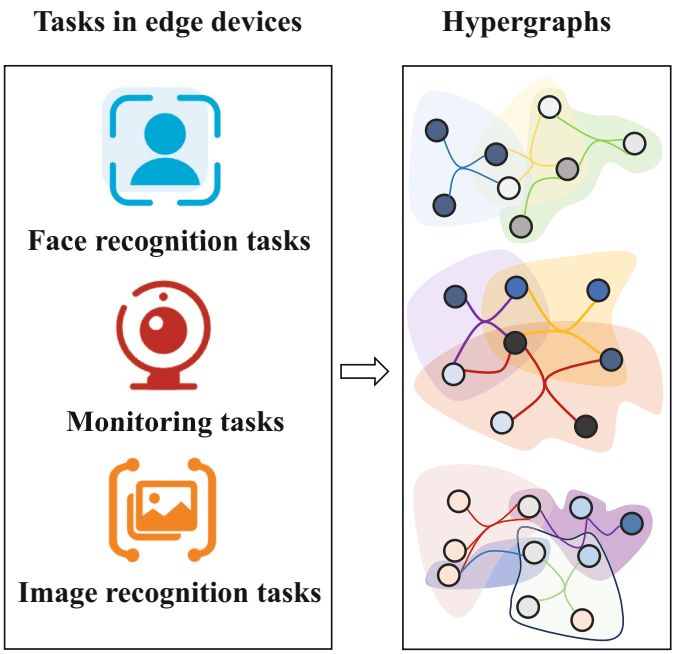

Fig. 1. The hypergraph representing different tasks.

3.2 Resource Pool Model in CPN

The resource pool in CPN can be represented as a collection of heterogeneous CPN nodes formed through virtualization of computing resources, denoted by $\mathcal{U} = \{0, 1, 2, \ldots, u, \ldots, |\mathcal{U}|\}$. For convenience, we model the local servers of the ED system as nodes similar to CPN nodes, and we denote the ED system as node 0. To cater to various services, it is assumed that each CPN node is equipped with k virtual machines (VMs), with each VM responsible for processing different types of tasks and exhibiting unique computational capabilities. $CP_u = \{CP_{u,v} | v \in \mathcal{V}\}$ denotes the task processing capabilities of CPN node u for various task types. Here, $CP_{u,v}$ quantifies the combined computing capability of the u-th CPN node when managing tasks of type v. These tasks can be either executed locally or offloaded to a resource pool, where services with matching performance levels are accessible. Furthermore, the CPN system state contains the transmission rate of wireless uplink channel r_{ul}, and rate of downlink channel, r_{dl}, which is similar to the MEC system [21].

3.3 Latency Calculation Model

In the scheduling of $\mathcal{H} = (\mathcal{T}, \mathcal{E})$ the task offloading policy $A_{1:n} = \{a_i | i \in \{1, 2, \ldots, n\}\}$ is a critical factor. Each task t_i has an offloading decision value $a_i \in \mathcal{U}$, 0 indicates running the object detection task on ED; otherwise, it will be offloaded to the corresponding CPN node. It's worth noting that when a directed hyperedge with an $eType_j$ of 0 connects multiple tasks, the policies of these tasks should be the same.

Each task will be offloaded onto VMs tailored to handle tasks of their respective types. For task t_i, The overall duration $T_{i,u}$ required for offloading task t_i to CPN node u is comprised of the following components: $T^{dt}_{i,u}$, which denotes the time taken for data transmission:

$$T^{dt}_{i,u} = \frac{inData_i}{r_{ul}} + \frac{outData_i}{r_{dl}}, u > 0, \tag{2}$$

which contains the time for uploading data as well as receiving data; $T^{ex}_{i,u}$ denotes the execution time of the task on the CPN node or the ED system:

$$T^{ex}_{i,u} = \frac{C_i}{CP_{u,type_i}}, \tag{3}$$

when u equals 0, it signifies the execution of tasks in the ED system.

The directed hypergraph structures we discuss are all acyclic, allowing us to represent tasks as sequences through weighted topological sorting. We replace the queues in the topological sort with min-heaps, using the task's earliest completion time as the key:

$$CT_i = \max_{k \in pre(i)} (CT_k + T_{k,a_j}), \tag{4}$$

where $pre(\cdot)$ represents all predecessor tasks of a task:

$$pre(t_i) = \{k | k \in pred_j, t_j = t_i, j \in \{1, 2, \ldots, m\}\}. \tag{5}$$

When employed in the task allocation process, this sequence guarantees strict compliance with dependency constraints. As illustrated in Fig. 2, the tasks within the hypergraph are parsed into sequences and continuously allocated to their respective resource pools through the offloading policy.

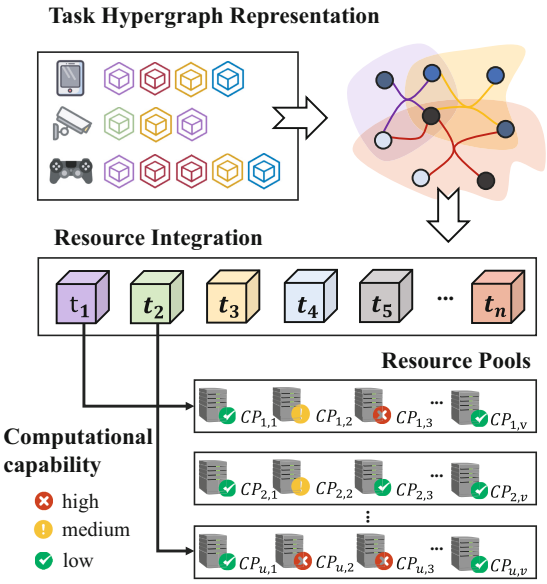

Fig. 2. Task offloading in CPNs.

VM on each CPN node and the ED system can exist in two states: one is a waiting state with no running tasks, and the other is a running state with an active task. Waiting VMs can transition to a running state immediately upon task offloading, while running VMs can return to a waiting state after task completion. We use $ST_{u,v}$ to denote the state of the v-th task type running on the u-th CPN node's VM, where 0 represents a waiting state, and 1 represents a running state. $AT_{u,v}$ represents the available time of a VM, which depends on its current state and the task it is running:

$$AT_{u,v} = \begin{cases} currentTime, & \text{if } ST_{u,v} == 0, \\ WAT_{u,v} + T^{ex}_{i,u}, & \text{if } u == 0 \text{ and } ST_{u,v} == 1, \\ WAT_{u,v} + T_{i,u}, & \text{else}, \end{cases} \tag{6}$$

where $WAT_{u,v}$ equals the time at which it was last in the waiting state. It is worth noting that when offloading to the ED system, there is no need to consider

data transmission time. Due to the possibility of multiple tasks being offloaded onto the same VM, we need to establish a waiting queue q_u. Tasks offloading at the same time are queued based on their position in the task sequence. Consequently, the task t_i upon allocation to a VM, receives a position in the waiting queue, denoted as $p_i = \{0, 1, \ldots, qlen\}$ with 0 indicating that the task is currently running, and $qlen$ representing the length of the waiting queue. Under normal circumstances, task t_i can estimate a projected finish time PFT_i:

$$PFT_i = AT_{u,type_i} + \sum_{j=1}^{p_i-1} T_{q_{u,j},u}. \tag{7}$$

If task t_i is executed in the ED system, its completion time is equal to:

$$PFT_i = AT_{0,type_i} + \sum_{j=1}^{p_i-1} T^{ex}_{q_{0,j},0}. \tag{8}$$

t_i must wait for its parent task to complete before starting allocation, the earliest start time EST_i is:

$$EST_i = \max_{k \in pre(i)} PFT_k + T^R_{k,i}, \tag{9}$$

where

$$T^R_{k,i} = \{eW_j | k \in pred_j, t_j = t_i, j \in \{1, 2, \ldots, m\}\}. \tag{10}$$

With the knowledge of EST_i and the offloading policy a_i, t_i can determine the position of the waiting queue on the VM of the corresponding CPN node. Iterating continuously allows us to ultimately obtain the expected completion times for all nodes. Since the exit task always lags behind all tasks, the earliest start time for the exit task is equal to the total time for completing all allocations:

$$T_{A_{1:n}} = EST_{t_e}. \tag{11}$$

3.4 Energy Consumption Calculation Model

Sending data to the resource pool via the uplink and downlink channels results in energy expenditure. The energy cost of offloading the task t_i to the u-th CPN node in the resource pool can be expressed as:

$$E^{dt}_{i,u} = TP_{i,u} T^{dt}_{i,u}, \tag{12}$$

where $TP^{dt}_{i,u}$ represents the transmission power of the uplink and downlink wireless channels. Assuming that the resource pool has different computing capacities and employs parallel computing through virtualization, allowing independent computation on each CPN node, the energy consumption of task t_i offloaded to u-th CPN node can be expressed as follows:

$$E^{ex}_{i,u} = \kappa^M_u (CP_{u,v})^2 C_i. \tag{13}$$

The effective switch capacitance of u-th CPN node is denoted by κ_u^M [26]. Therefore, the total energy consumption of offloading task t_i to CPN node u in the resource pool can be represented as follows:

$$E_{i,u} = E_{i,u}^{ex} + E_{i,u}^{dt}. \tag{14}$$

Therefore, according to the offloading policy $A_{1:n}$, the total energy consumption can be represented as follows:

$$E_{A_{1:n}} = \sum_{i=1}^{n} E_{i,a_i}. \tag{15}$$

3.5 Problem Formulation

The main objective of task offloading is to determine an efficient offloading policy to optimize the total latency and total energy consumption. Formally, based on the aforementioned computation model, the optimization objective can be formulated as follows:

$$(\mathbf{P}) \quad \min_{A_{1:n}} \omega_t T_{A_{1:n}} + \omega_e E_{A_{1:n}}, \tag{16}$$

where ω_t and ω_e represent weighted parameters set by the CPNs system, which signify the significance assigned to the optimization objectives.

4 The Dat-DRL Solution

To tackle the task offloading optimization issue in CPNs, we introduce a novel DRL algorithm, namely Dat-DRL. By leveraging the PPO algorithm along with dual attention mechanisms, we can more effectively capture the interdependencies between tasks and diverse service requirements. Furthermore, this algorithm significantly reduces energy consumption.

4.1 Markov Decision Process

In the realm of reinforcement learning, the task offloading process can be effectively modeled as a markov decision process (MDP). Within a given task sequence $\mathcal{T}$, the MDP associated with each learning task comprises the state space $\mathcal{S}$, action space $\mathcal{A}$, and reward function $\mathcal{R}$, which are defined as follows:

– **State space $\mathcal{S}$**: We execute offloadig for the task sequence one by one, where the MDP state of offloading the i-th task can be represented as a hypergraph embedding along with the offloading policies for the previous i tasks:

$$s_i = (\mathcal{EH}(\mathcal{T}, \mathcal{E}), A_{1:i}), \tag{17}$$

where $\mathcal{EH}(\mathcal{T}, \mathcal{E})$ comprises task embeddings.

- **Action space** $\mathcal{A}$: For each task t_i, the action $a_i \in \mathcal{U}$ has different interpretations: 0 denotes local execution on the ED system, while other values correspond to CPN nodes in the resource pool.
- **Reward function** $\mathcal{R}$: In order to minimize the issues in (16), the reward function is designed to resemble the negative growth in (16). Specifically, when task t_i takes action a_i, the increment is defined as follows:

$$r_i = -(\omega_t T_{A_{1:i}} + \omega_e E_{A_{1:i}} - (\omega_t T_{A_{1:i-1}} + \omega_e E_{A_{1:i-1}})). \tag{18}$$

  Therefore, the reward function encourages tasks to make offloading decisions that improve system performance.
- **Policy function** π: Given the aforementioned MDP definition, the offloading policy function for task t_i can be represented as $\pi(a_i|s_i)$. For a hypergraph containing n tasks, the probability of an offloading policy $A_{1:n}$ can be expressed as $\pi(A_{1:n})$. By applying the probability chain rule to each $\pi(a_i|s_i)$, we can obtain the overall probability as follows:

$$\pi(A_{1:n}) = \prod_{i=1}^{n} \pi(a_i|s_i). \tag{19}$$

After constructing the MDP, the task embeddings of the hypergraph $\mathcal{H}(\mathcal{T}, \mathcal{E})$ are used as inputs to the model. The model's output consists of the decisions made for each task and their corresponding state-value functions.

4.2 Dat-DRL Network

Figure 3 illustrates the architecture of the seq2seq neural network with dual attention mechanisms, specifically designed for CPNs task offloading. The specific computation process is as follows: Firstly, it is necessary to sample a trajectory of task offloading using the previous old policy and calculate the embedding information $\mathcal{TE} = \{te_1, te_2, \ldots, te_n\}$ by using the task attributes and adjacency matrix of the hypergraph for each task sequence, which is then input into a neural network. The network employs a seq2seq architecture, consisting of an encoder and a decoder. Before feeding the task into the encoder, we utilize a self-attention mechanism [18] to transform the task embedding into a self-attention encoding $\mathcal{SAE} = \{sa_1, sa_2, \ldots, sa_n\}$ that captures dependencies between tasks and task requirements:

$$sa_i = \sum_{j=1}^{n} value_j \alpha_{i,j}, \tag{20}$$

where

$$\alpha_{i,j} = softmax(\sum_{k=1}^{n} \frac{\exp(sim(i,j)}{\exp(sim(i,k))}), \tag{21}$$

$$sim(i,j) = key_i value_j. \tag{22}$$

In this context, $query_i, value_i$, and key_i are three vectors generated through fully connected layers based on task embedding vector te_i. $sim(i,j)$ is used to

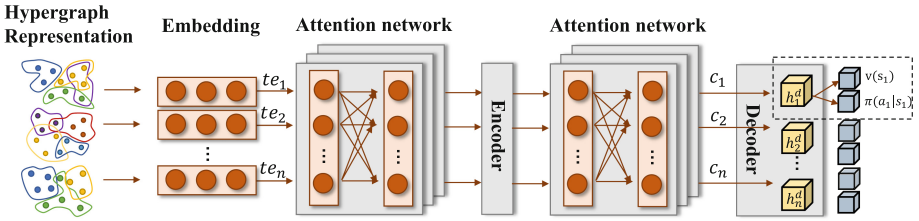

Fig. 3. Seq2Seq network architecture based on dual attention mechanism.

represent the correlation between i-th task and j-th task, which helps us better capture the relationships between tasks.

Then, the self-attention encoding is fed into the encoder, with the encoder utilizing LSTM networks. The LSTM network enables the computation of the encoder's hidden state h_i^e corresponding to each task. The final hidden state is used as the context vector input to the decoder as the first hidden state, which also employs an LSTM. We use an attention mechanism to optimize the computation of the memory unit c and the hidden state h_i^d:

$$c_i = \sum_{j=1}^{n} \alpha_{i,j}^d h_j^e, \tag{23}$$

where

$$\alpha_{i,j}^d = softmax(\sum_{k=1}^{n} \frac{\exp(cos(i,j)}{\exp(cos(i,k))}), \tag{24}$$

$$cos(i,j) = \frac{h_{i-1}^d h_j^e}{|h_{i-1}^d||h_j^e|}. \tag{25}$$

Of significance, cosine similarity is employed in the decoder's attention calculation to effectively capture directional information. Subsequently, the current hidden state h_i is computed through a memory unit c_i, the previous hidden state h_{i-1}^d, and the previous action a_{i-1}:

$$h_i = f(c_i, h_{i-1}^d, a_{i-1}), \tag{26}$$

where $f(\cdot)$ is a non-linear network with an activation function. The final offloading policy function $\pi(a_i|s_i)$ and value function $v(s_i)$ are computed from the decoder's hidden states through fully connected layers. For n tasks, the primary time consumed during each training iteration is attributed to the calculations within the fully connected layers and attention networks. Therefore, the training time complexity for each hypergraph is $O(n^2)$, which is similar to the attention-based seq2seq neural network [19]. The dual attention-based neural network architecture enables efficient task offloading decisions in dynamic CPNs, thereby enhancing resource utilization and system performance.

In order to achieve superior performance in task offloading, the PPO algorithm [17] was chosen as the fundamental RL technique in this study. PPO enables trajectory generation for individual learning tasks t_i based on the old policy $\pi(a_i|s_i;\theta_i^o)$, and allows for the update of the new policy $\pi(a_i|s_i;\theta_i)$ across multiple iterations. Here, θ_i^o and θ_i represent the previous policy parameters and the new parameters that necessitate updating. To ensure the stability of the learning process and prevent excessively large updates, the total reward of tasks is clipped using the following form:

$$\mathcal{L}_i^{\mathrm{CLIP}}(\theta_i) = \mathbb{E}_{\tau\sim pdf_i(\tau,\theta_i^o)}\left[\sum_{j=0}^{n}\min(r(\theta_j,\theta_j^o)\hat{A}_j, clip_{1-\epsilon}^{1+\epsilon}(r(\theta_j,\theta_j^o))\hat{A}_j\right], \quad (27)$$

where $pdf_i(\tau,\theta_i^o)$ denotes the probability distribution of the sampled trajectory τ generated based on the old sample policy function $\pi(a_i|s_i;\theta_i^o)$. The probability distribution ratio that exists between the sample policy and the target policy is represented by $r(\theta_j,\theta_j^o)$ and is defined as follows:

$$r(\theta_j,\theta_j^o) = \frac{\pi(a_j|s_j;\theta_j)}{\pi(a_j|s_j;\theta_j^o)}. \quad (28)$$

The term $clip_{1-\epsilon}^{1+\epsilon}(r_j(\theta_j s,\theta_j^o))$ is used to constrain the probability distribution ratio within the range $[1-\epsilon,1+\epsilon]$. Here, $\hat{A}_j$ represents the estimated advantage function, which approximates the difference between the expected reward and the actual reward for the trajectory after the task t_i takes the corresponding action. $\hat{A}_j$ is a generalized advantage estimation (GAE) function [16] commonly used in reinforcement learning:

$$\hat{A}_j = \sum_{k=0}^{n-j+1}(\gamma\lambda)^l(r_{j+k} + \gamma v_\pi(s_{j+k+1}) - v_\pi(s_{j+k})), \quad (29)$$

where $\gamma \in [0,1]$ is the discount factor used to calculate the cumulative error, and λ is a parameter used to control bias and variance. The loss function of the state-value function is defined as follows:

$$\mathcal{L}_i^{\mathrm{MSE}}(\theta_i) = \mathbb{E}_{\tau\sim P_{t_i}(\tau,\theta_i^S)}\left[\sum_{j=0}^{n}(v_\pi(s_j) - \hat{v}_\pi(s_j))^2\right], \quad (30)$$

where $\hat{v}_\pi(s_j) = \sum_{k=0}^{n-j+1}\lambda^k r_{j+k}$, The loss function for each task can be derived using Eqs. (27) and (30):

$$\mathcal{L}_i(\theta_i) = \mathcal{L}_i^{\mathrm{CLIP}}(\theta_i) + \mathcal{L}_i^{\mathrm{MSE}}(\theta_i). \quad (31)$$

The training process continuously optimizes the offloading policy through parameter gradient ascent:

$$\hat{\theta}_i = \theta_i + \omega\sum_{j=1}^{z}\nabla_{\theta_i}\mathcal{L}_i(\theta_i), \quad (32)$$

where ω represents the learning rate for gradient ascent, and $\nabla_{\theta_i}\mathcal{L}_i(\theta_i)$ denotes the computed gradient, which is updated after z steps to replace the old sample policy in PPO. To tackle the computational cost and implementation challenges arising from complex neural network combinations, we employ the Adam first-order [6] approximation for gradient computation.

5 Experiments

5.1 Hyperparameters Setting

We employ TensorFlow, an open-source framework for constructing and training deep learning models, to implement the DRL algorithm. TensorFlow enables automatic gradient computation for variables. The standard training process involves the use of several hyperparameters, which are presented in the Table 1.

Table 1. The training hyperparameters.

Hyperparameter	Value	Hyperparameter	Value
Batch size	20	Reinforcement learning algorithm	PPO
Encoder and decoder layers	2	Encoder and decoder layers type	LSTM
Hidden units	128	Encoder and decoder layers norm	On
Learning rate ω	2e-4	Gradient steps z	4
Activation function	Tanh	Optimization method	Adam
Clipping constant	0.2	Weighted factor c_1	0.5
Discount factor γ	0.99	Advantage function discount factor λ	0.95

5.2 Simulation Environment Configuration

We establishe a virtual environment for simulating task offloading and implements reinforcement learning through interaction with the environment. To characterize heterogeneous directed hypergraphs, we utilize a synthetic directed acyclic graph (DAG) generator [21] for simulating computationally intensive tasks and applied hypergraph theory to transform them into hypergraphs [9]. The standard virtual environment utilizes various detailed parameters, as shown in the Table 2.

5.3 Experimental Results

Training Performance Comparison. In order to evaluate the effectiveness of Dat-DRL, we perform a comparative analysis with various learning algorithms, including: (1) TRPO algorithm [15], (2) PPO algorithm [17] without attention mechanism, (3) PPO algorithm with a single attention mechanism [1](A-PPO), the single layer of attention is utilized to compute the hidden units within the

Table 2. The environment parameters.

Parameter	Value
ED computational capacity $CP_{0,v}$	[1 GHz, 2 GHz]
Number of the CPN node's VM	4
Number of CPN nodes	4
Number of the tasks	[5, 45]
Number of the hyperedges	[20, 60]
Task data size	[5 Kb, 10 Mb]
Computational capability requirements C_i	[2 GHz, 15 GHz]
VM computational capacity $CP_{u,v}$	[2 GHz, 20 GHz]
Transmission rate $r_{i,u}$	[1.0 Mbps, 10.0 Mbps]

decoder and is also the most commonly used attention-based structure in seq2seq network architectures, (4) Dat-DRL which is proposed in this paper.

Figure 4 illustrates the comparison of learning performance among the aforementioned algorithms, clearly indicating the superiority of our proposed Dat-DRL algorithm. Compared to the regular PPO algorithm without an attention mechanism, Dat-DRL demonstrates a substantial improvement of approximately 20.08% in the system's objective reward. Moreover, when compared to the TRPO algorithm, Dat-DRL achieves an even higher enhancement of 31.98% in the system's objective, highlighting its superior learning capability over the other two algorithms. In addition to the remarkable performance in the system's objective, Dat-DRL exhibits a significantly accelerated convergence speed of 53.40% compared to the traditional attention-augmented Seq2Seq network. This accelerated convergence speed translates into substantial savings in training time and costs, proving to be highly beneficial for power transmission line defect perception.

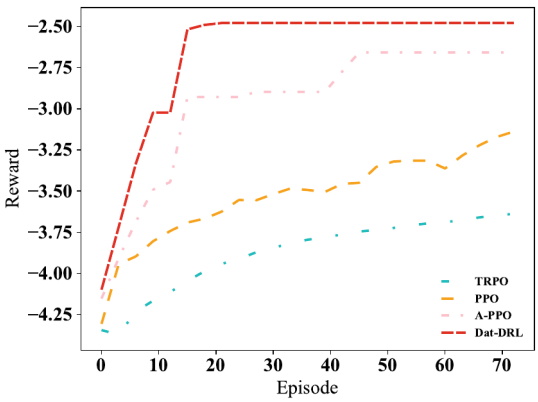

Fig. 4. Training results of different learning algorithms.

Results with Various Environment. To evaluate the effectiveness and adaptability of the algorithm, we compare it with six other baseline algorithms in task allocation performance. These algorithms include:

1. Coarse-grained Remote Offloading (Remote): Tasks are considered as a whole without considering their finer granularity, and all tasks are offloaded to the remote resource pool for execution.
2. Coarse-grained Local Execution (Local): Tasks are considered as a whole without considering their finer granularity, and all tasks are executed locally on the ED system.
3. Random: Tasks are randomly assigned to be executed either locally or in the remote resource pool.
4. Greedy: This algorithm considers only the latency of tasks. Each task is offloaded to the resource pool or executed locally based on the shortest waiting time and execution time.
5. TRPO: It considers KL-divergence as a penalty term in gradient updates to ensure learning stability.
6. PPO without attention mechanism: It employs gradient clipping to limit gradient updates, enabling rapid policy learning and demonstrating good performance.

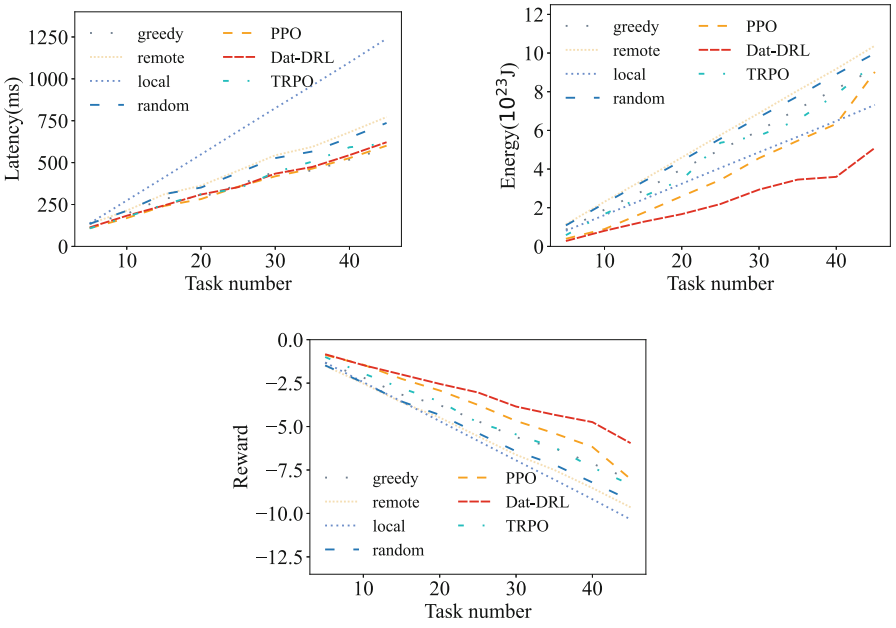

Fig. 5. Results with different number of tasks.

Figure 5 investigates the impact of task quantity on latency, energy consumption, and system objective. The results reveal that an increase in task numbers leads to higher latency and energy consumption during task execution and data transmission. Consequently, all algorithms experience increased computational latency, energy usage, and a corresponding drop in system objective (where the system objective represents the normalized negative value of latency and energy consumption). It's worth noting that the greedy algorithm exhibits remarkably low latency because it only considers task latency. However, even when considering energy consumption, Dat-DRL also demonstrates similarly low latency. In contrast, a significant enhancement in energy efficiency is observed in Dat-DRL compared to the greedy algorithm, without compromising task latency. This underscores Dat-DRL's ability to minimize energy consumption while maintaining low latency. Consequently, Dat-DRL achieves the highest overall objective reward among all algorithms, emphasizing its effectiveness in optimizing both latency and energy consumption concurrently.

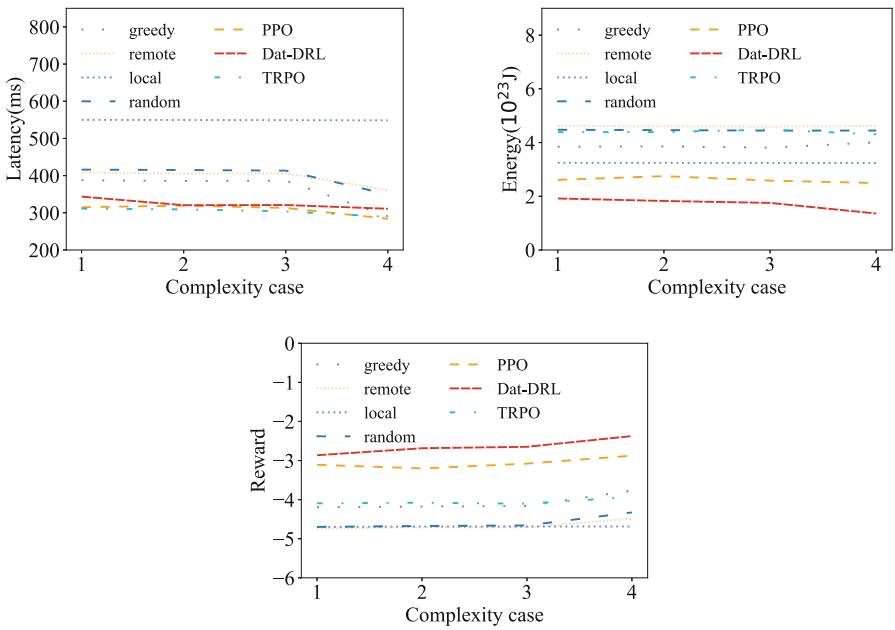

Fig. 6. Results with different topological structures of the hypergraph.

Figure 6 presents Dat-DRL's results and corresponding baselines for latency, energy consumption, and system objective across four distinct directed hypergraphs with varying topological structures. The X-axis of the experimental result plots represents increasing complexity in the four different topological structures. Hypergraph topology complexity primarily depends on density and the number of hyperedges, where higher density and more hyperedges signify a stricter task

priority order. Dat-DRL attains the optimal reward objective, striking the best balance between latency and energy consumption for various topological structures. Concerning latency, Dat-DRL closely trails the Greedy algorithm and TRPO, with marginal differences. However, in terms of energy consumption, Dat-DRL holds a significant advantage over other algorithms.

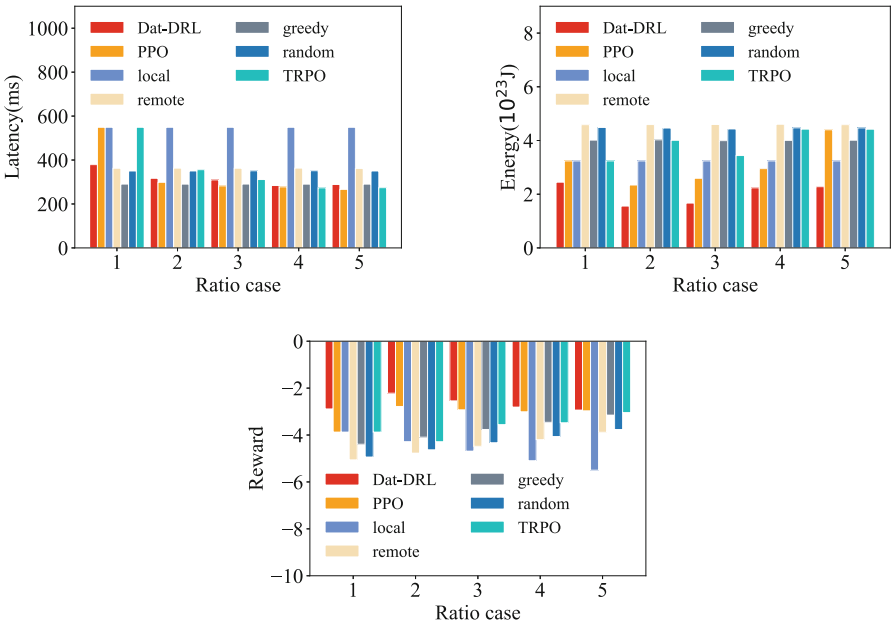

Fig. 7. Results with different weighted parameters.

Figure 7 illustrates experiments conducted under five distinct scenarios with varying latency and energy consumption requirements, including: (1) $\omega_t : \omega_e = (0.1, 0.9)$, (2) $\omega_t : \omega_e = (0.3, 0.7)$, (3) $\omega_t : \omega_e = (0.5, 0.5)$, (4) $\omega_t : \omega_e = (0.7, 0.3)$, and (5) $\omega_t : \omega_e = (0.9, 0.1)$. The first two scenarios lean towards reducing energy consumption, while the fourth and fifth scenarios prioritize latency considerations. Experimental results show that Dat-DRL achieves the optimal trade-off between energy consumption and latency, outperforming baseline algorithms. Specifically, compared to remote, local, random, greedy, TRPO, and PPO algorithms in different scenarios, Dat-DRL reduces average energy consumption by 14.81%, 43.79%, 12.06%, −6.24%, 12.54%, and 7.85%, respectively. In terms of energy consumption, Dat-DRL outperforms baseline algorithms by reducing it by 61.29%, 45.03%, 60.02%, 55.56%, 54.40%, and 42.64%, respectively.

In task offloading scenarios at edge devices on power transmission lines, network conditions are complex, dynamic, and critical. The experiments encompass five different network conditions, and Fig. 8 compares latency, energy consumption, and system objective among different algorithms under varying

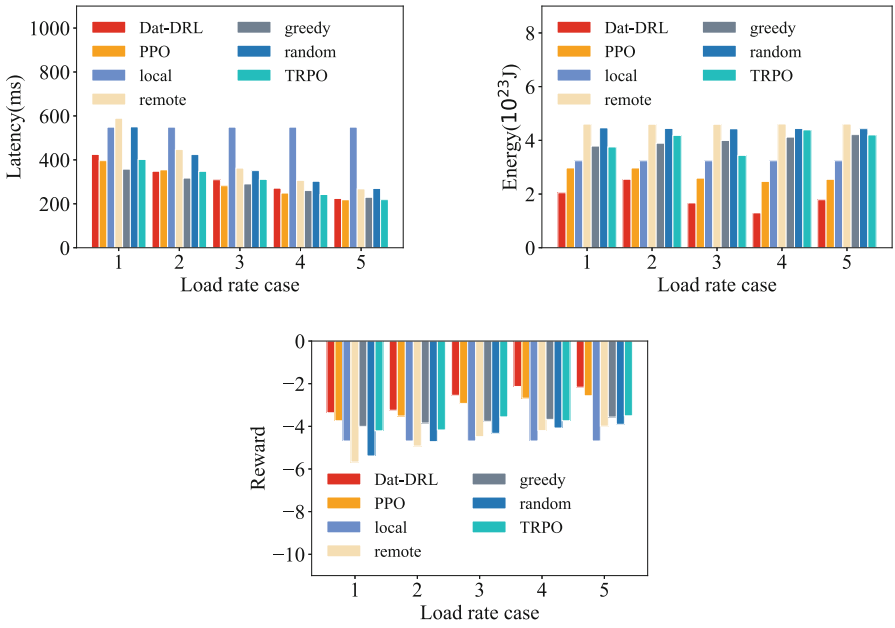

Fig. 8. Results with different network conditions.

transmission rates. The transmission rates gradually increase in the five scenarios. The Dat-DRL algorithm intelligently coordinates offloading strategies, achieving latency and energy consumption close to optimal when compared to other algorithms. This underscores Dat-DRL's capability to efficiently manage dynamic task offloading scenarios with varying transmission rates, highlighting its effectiveness in handling diverse network conditions.

6 Conclusion

We present a novel task offloading algorithm to address insufficient computing power in edge devices for object detection. Initially, we establish a simulated environment for task offloading scenarios, modeling tasks as directed hypergraphs. We provide computational formulas for optimizing task offloading in terms of latency and energy consumption objectives. Furthermore, we simulate a realistic configuration of heterogeneous resource pools, integrating dispersed resources into fine-grained resource pools, each equipped with specialized computing resources for different task types. Subsequently, we introduce the DRL algorithm, Dat-DRL, which utilizes a Seq2Seq network structure with a dual-attention mechanism to enhance offloading strategies by capturing diverse inter-task dependencies more effectively. To validate the effectiveness and adaptability of Dat-DRL, we conduct comprehensive experiments, comparing the results

across various environments and benchmarking against other baseline algorithms for task offloading. These experiments demonstrate the efficacy of Dat-DRL in reducing energy consumption.

Acknowledgment. The work is funded by the science and technology project of SGCC (State Grid Corporation of China): Research on Key Technologies and Applications of Intelligent Edge Computing for Transmission Line Defect Sensing (5700-202318309A-1-1-ZN).

References

1. Chai, F., Zhang, Q., Yao, H., Xin, X., Gao, R., Guizani, M.: Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite IoT. IEEE Trans. Veh. Technol. **72**(6), 7783–7795 (2023)
2. Chen, J., Yang, Y., Wang, C., Zhang, H., Qiu, C., Wang, X.: Multitask offloading strategy optimization based on directed acyclic graphs for edge computing. IEEE Internet Things J. **9**(12), 9367–9378 (2021)
3. Chen, Y., Zhang, N., Zhang, Y., Chen, X., Wu, W., Shen, X.: Energy efficient dynamic offloading in mobile edge computing for internet of things. IEEE Trans. Cloud Comput. **9**(3), 1050–1060 (2019)
4. Di, Z., et al.: In-network pooling: contribution-aware allocation optimization for computing power network in b5g/6g era. IEEE Trans. Netw. Sci. Eng. **10**(3), 1190–1202 (2022)
5. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
6. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
7. Kumar, A., Zhou, A., Tucker, G., Levine, S.: Conservative q-learning for offline reinforcement learning. Adv. Neural. Inf. Process. Syst. **33**, 1179–1191 (2020)
8. Lei, L., Xu, H., Xiong, X., Zheng, K., Xiang, W.: Joint computation offloading and multiuser scheduling using approximate dynamic programming in NB-IoT edge computing system. IEEE Internet Things J. **6**(3), 5345–5362 (2019)
9. Li, C., Zhang, Y., Hao, Z., Luo, Y.: An effective scheduling strategy based on hypergraph partition in geographically distributed datacenters. Comput. Netw. **170**, 107096 (2020)
10. Liu, Y., Wei, X., Xiao, J., Liu, Z., Xu, Y., Tian, Y.: Energy consumption and emission mitigation prediction based on data center traffic and PUE for global data centers. Global Energy Interconnect. **3**(3), 272–282 (2020)
11. Ming, Z., Li, X., Sun, C., Fan, Q., Wang, X., Leung, V.C.: Dependency-aware hybrid task offloading in mobile edge computing networks. In: 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS), pp. 225–232. IEEE (2021)
12. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937. PMLR (2016)
13. Ning, Z., Dong, P., Kong, X., Xia, F.: A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things. IEEE Internet Things J. **6**(3), 4804–4814 (2018)
14. Ren, X., et al.: AI-bazaar: a cloud-edge computing power trading framework for ubiquitous AI services. IEEE Trans. Cloud Comput. **11**(3), 2337–2348 (2022)

15. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897. PMLR (2015)
16. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438 (2015)
17. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
18. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
19. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Advances in Neural Information Processing Systems, vol. 28 (2015)
20. Wang, J., Hu, J., Min, G., Zhan, W., Ni, Q., Georgalas, N.: Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. IEEE Commun. Mag. **57**(5), 64–69 (2019)
21. Wang, J., Hu, J., Min, G., Zomaya, A.Y., Georgalas, N.: Fast adaptive task offloading in edge computing based on meta reinforcement learning. IEEE Trans. Parallel Distrib. Syst. **32**(1), 242–253 (2020)
22. Wang, X., Ren, X., Qiu, C., Xiong, Z., Yao, H., Leung, V.C.: Integrating edge intelligence and blockchain: what, why, and how. IEEE Commun. Surv. Tutor. **24**(4), 2193–2229 (2022)
23. Wu, F., et al.: Energy-time efficient task offloading for mobile edge computing in hot-spot scenarios, pp. 1–6 (2021)
24. Zhai, Y., et al.: An energy aware offloading scheme for interdependent applications in software-defined IoV with fog computing architecture. IEEE Trans. Intell. Transp. Syst. **22**(6), 3813–3823 (2020)
25. Zhao, M., et al.: Energy-aware task offloading and resource allocation for time-sensitive services in mobile edge computing systems. IEEE Trans. Veh. Technol. **70**(10), 10925–10940 (2021)
26. Zhou, X., Zhao, J., Han, H., Guet, C.: Joint optimization of energy consumption and completion time in federated learning. In: 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), pp. 1005–1017. IEEE (2022)