DNN Inference Task Offloading Based on Distributed Soft Actor-Critic in Mobile Edge Computing

Wenxiu Xu
College of Computer and Electronic Information
Guangxi University
Nanning, China
xwx_6838@163.com

Ningjiang Chen*
College of Computer and Electronic Information
Guangxi University
Nanning, China
chnj@gxu.edu.cn

Huan Tu
College of Computer and Electronic Information
Guangxi University
Nanning, China
th_1998@163.com

***Abstract*—In mobile edge computing, DNN-driven intelligent inference service is highly sensitive to latency. Recently, collaborative inference between user devices and Edge Servers (ESs) based on DNN partition has been used in service acceleration. However, due to the limited computing resources of ESs, there is resource competition between concurrent requests, resulting in the partition tasks cannot be offloaded to ESs in time. Therefore, it is necessary to design an efficient offloading scheme for partition-based concurrent inference tasks. Existing task offloading schemes based on Deep Reinforcement Learning (DRL) can solve complex decision-making problems in high-dimensional state space, but there are problems such as insufficient sample diversity and easily falling into local optimum. Therefore, we propose a collaborative DNN inference task offloading scheme based on distributed Soft Actor-Critic(SAC). It supports SAC Agents to explore samples in parallel and share learning experiences, and improves the randomness of the policy through the maximum entropy mechanism to avoid falling into local optimum, thus achieving efficient offloading of concurrent partition tasks. Experimental results on DNN benchmarks show that compared with the baseline schemes, the average service latency of our scheme is reduced by more than 18.3%, and it has a higher convergence speed and task success rate, which can make ESs achieve load balancing.**

Keywords-component; mobile edge computing; DNN inference; task offloading; distributed SAC; experience sharing

I. INTRODUCTION

Edge intelligent inference services driven by Deep Neural Networks (DNN) are rapidly spreading on Internet of Things (IoT) devices [1], such as image recognition, video processing, and augmented reality, which are highly sensitive to latency. The traditional method uses the powerful computing power of the cloud computing center to provide low-latency DNN inference services [2], but the long-distance transmission of media data will generate high transmission latency and energy consumption, while mobile edge computing (MEC) will computing resources sink to the edge near the data source, providing users with more agile service response by deploying Edge Servers (ESs) [3].

Recently, collaborative inference between User Devices (UDs) and ESs based on DNN partitioning in MEC has been widely used in service acceleration [4], because UDs can efficiently process the frontend part of DNN requests, greatly reducing data transmission latency. However, the computing resources of ESs are limited, and there is resource competition between partition-based concurrent inference tasks, which may lead to uneven task allocation between ESs and even ESs overload, thus failing to achieve the acceleration effect of collaborative inference. Therefore, how to offload partition-based concurrent inference tasks to ESs with limited resources to reduce service latency and achieve load balancing among ESs has become an urgent problem to be solved.

Recent task offloading methods model the offloading process as a Markov Decision Process (MDP), and use Deep Reinforcement Learning (DRL) technology to solve the MDP problem, and then offload the tasks to appropriate ESs [5], which reduces the service latency in a MEC environment with limited resources. Because DRL can effectively exert the feature extraction ability of deep learning and the learning ability of reinforcement learning and solve the complex decision-making problems in high-dimensional state space [2]. For example, Liu et al. [6] proposed a task offloading algorithm based on Dueling Deep Q-Network (DDQN), which realized the online task offloading for service acceleration under stochastic task generation and dynamic network conditions. Wu et al. [7] modeled the offloading problem as a constrained MDP and proposed an inference task offloading algorithm based on Deep Deterministic Policy Gradient (DDPG) by using the Lyapunov optimization technique, which realized the optimal allocation of computing resources. However, most of the existing DRL-based task offloading schemes use centralized Agent exploration to continuously interact with the environment [5], which has the problems of insufficient diversity of learning experience and high exploration cost. Concurrently, Agent has low exploration efficiency and sample learning rate in the exploration process, which makes the policy difficult to converge and easy to fall into local optimum.

Therefore, to solve the problem of partition task offloading in a high concurrent MEC environment, we propose a collaborative DNN inference task offloading scheme CDO-DSAC based on distributed Soft Actor-Critic (SAC), which determines the optimal offloading decision for a set of partitioning-based concurrent inference tasks. The main contributions of this paper are as follows:

- We model the offloading problem as MDP with entropy and propose CDO-DSAC to solve it. CDO-DSAC supports SAC Agents to explore in parallel to share learning experiences for policy optimization, and periodically selects the Agent with the highest average return to update the optimal policy parameters synchronously, which solves the problems of insufficient diversity of learning experience and high cost of Agents exploration in centralized training.

DOI reference number: 10.18293/SEKE2023-150

This work is supported by the National Natural Science Foundation of China (No.62162003), and the Nanning Science and Technology project (No. 20221031).

- CDO-DSAC takes the maximum entropy as the goal to improve the randomness of the policy, to avoid the policy falling into the local optimum, and obtain the offloading decision with better latency. Concurrently, it encourages Agents to explore through the automatic entropy adjustment mechanism to improve their sample learning rate and convergence speed. The experimental results show that compared with the baseline schemes, CDO-DSAC has better performance in terms of acceleration performance and reliability, and has higher convergence speed and average return, which effectively reduces the exploration cost.

II. Related Work

To achieve load balancing between ESs, the method based on task offloading offloads computing tasks to appropriate ESs accelerates task execution while improving resource utilization. Some studies have used traditional heuristic methods based on linear/nonlinear optimization, genetic algorithm, and game theory to achieve task offloading in MEC, and achieved good service acceleration results. For example, Chen et al. [8]modeled the task offloading problem in MEC as a mixed integer nonlinear optimization problem and designed an efficient task offloading scheme SDTO. Literature [9] proposed a distributed computing offloading scheme based on a matching game mechanism, which offloads partition-based inference tasks to the edge cloud to achieve service acceleration. However, the above methods do not have sufficient autonomous decision-making capabilities and cannot achieve the expected acceleration performance in a dynamic MEC environment.

In recent years, RL and DRL technologies play a key role in solving the above problems. For example, Xu et al.[10] designed an RL-based inference task online admission algorithm Online RL, which generates an offloading strategy for randomly arrived tasks. However, RL technology cannot cope with the decision-making problem of high-dimensional state space and lacks versatility and fast adaptability. DRL has strong feature extraction ability and learning ability, which provides a solution for task offloading problems in high-dimensional state space. For example, Literature [11] implemented a real-time offloading program based on Asynchronous-Advantage-Actor-Critic(A3C) to solve the task offloading problem in MEC stochastic environment. Ren et al. Literature [12] proposed an offloading optimization algorithm based on Proximal Policy Optimization (PPO) to solve the stochastic optimization problem of when and where tasks are offloaded.

Recent studies have applied advanced DRL algorithms to solve the task offloading problem. For example, Wu et al. [7] proposed a DDPG-based task offloading strategy to optimize resource allocation in continuous state space in the MEC environment. Literature [13] modeled the offloading problem as an MDP with constrained hybrid action space and proposed a DDPG-based offloading strategy D3PG. It optimizes computational offloading in a dynamic environment by joint task partitioning and computing power allocation. DDPG is a DRL algorithm with a deterministic policy gradient, which converges fast in continuous state space, but is not suitable for a stochastic environment.

However, the existing DRL-based task offloading schemes have two defects. First, centralized agent exploration does not consider the distributed characteristics of MEC, and there are problems of insufficient diversity of learning experience and high exploration cost. Second, in the process of policy training, there are problems of poor Agent exploration efficiency and low sample learning rate, which leads to difficult policy convergence and easy to falls into local optimum. This paper focuses on solving the above problems to improve the performance of partition-based concurrent inference task offloading in policy convergence and service acceleration while ensuring the reliability of offloading schemes in extreme MEC environments and load balancing between ESs.

III. Problem modeling

At time slot t, we define $I_e=\{I_{e,1}, I_{e,2},...I_{e,j}..., I_{e,n}\}$ as a set of partition-based concurrent inference tasks offloaded from UDs to ESs, and $\mathcal{E}=\{e_1, e_2,...e_i..., e_k\}$ is denoted as a set of ESs. To determine the optimal latency offloading policy, we formulate the offloading problem in the MEC network modeled as an MDP with an entropy term, where the four elements are defined as follows.

(1) State: At time slot t, the system state is denoted as $S^t=(I_e^t, \mathcal{E}^t, N^t)$, $I_e^t=\{I_{e,1}^t, I_{e,2}^t,...I_{e,j}^t..., I_{e,n}^t\}$ describes the state information of partition-based concurrent inference tasks; $\mathcal{E}^t=\{e_1^t, e_2^t,...e_i^t..., e_k^t\}$ describes the workload state of ESs, there is $e_i^t=(c_{i,new}^t, c_i^{max})$, where $c_{i,new}^t=c_i^{max}-c_i^{t-1}$ represents the current acceptable task calculation amount of e_i^t, determined by the maximum service capacity c_i^{max} and the task calculation loaded in the time slot t-1; N^t describes the network state, which means $N^t=(b^t, g^t)$, b^t is the network bandwidth, and g^t is the channel gain.

(2) Offloading Actions: in the policy exploration phase, each ES can be a candidate offloading action for an inference task, expressed as $a_j=\{e_1, e_2,...e_i..., e_k\}$, $e_i\in\{0,1\}$, and there is only one $e_i=1$. Therefore, at time slot t, the offloading actions of a set of tasks can be expressed as $A^t=(a_1, a_2,...a_j..., a_n)$.

(3) Reward function: once an offloading action is generated in the current state, the Agent will obtain a system instant reward from the environment, scoring the current offloading action A^t. The goal of collaborative offloading is to minimize the service latency of inference tasks, and the offloading actions of concurrent inference tasks will affect each other. Therefore, we define the reward as the negative value of the total service latency of the system under the offloading policy.

$$R^t(S^t,A^t)=-\sum_1^n T_j \; . \tag{1}$$

Because the DNN is divisible, we allow UDs to offload part of the inference tasks to the ESs, so the service latency T_j of the inference task is composed of the inference latency of the UDs side, the data transmission latency, the queuing latency and the inference latency of the ESs side.

(4) State-Action entropy: considering the influence of ESs workload state on the offloading action, the state-action entropy term $H(\pi(A^t\,|\,S^t))=E[-log\,\pi(A^t|\,S^t)]$ is added to improve the randomness of the policy while encouraging Agent exploration to avoid falling into local optimum, where $\pi(A^t|\,S^t)$ is the probability matrix of the offloading action A^t under S^t. Specifically, at time slot t, when multiple sets of offloading actions are optimal, the Agent will randomly select one, which ensures that each set of valuable offloading actions will not be ignored.

We define the behavior of generating offloading actions for a set of partition-based concurrent tasks as the collaborative offloading policy π_φ. The optimal offloading policy π_φ^* can be learned by maximizing the expectation of cumulative discount reward with entropy, that is to maximize the average return, denoted as:

$$\pi_\varphi^* = \arg\max_{\pi_\varphi} E[\sum_{t=1}^{\infty} \lambda^t (R^t + \alpha H(\pi(A^t|S^t)))], \quad (2)$$

where φ is the policy parameter, $\lambda^t \in [0,1)$ is the discounted factor, and α is the temperature coefficient that controls the randomness of the offloading policy.

IV. CDO-DSAC: Collaborative DNN Inference Task Offloading Based on Distributed SAC

A. *Overview and Workflow*

The overview and workflow of CDO-DSAC is shown in Figure 1, which consists of two parts. One part is distributed deployed on each ES, consisting of Communication Manager and SAC Agent. Communication Manager is responsible for communicating with UDs and ESs and collecting system state information, such as partition inference tasks status information, ESs workload state information, and network state information, and is responsible for offloading tasks to the application container instances of each ES according to the offloading decision, corresponding to steps ①, ② and ③. SAC Agent is a DRL network developed based on maximum entropy, which can approximate the optimal latency offloading policy according to the system state information, corresponding to ④. The other part is Centralized Controller deployed at the central node of the MEC network, which includes Shared-Experience Replay Memory D and Optimal Policy Updater. D is responsible for collecting the learning experiences, average return and policy parameter information explored by each Agent, corresponding to ⑤. Optimal Policy Updater is responsible for periodically selecting the SAC Agent with the largest average return as the optimal policy according to the information collected in D, corresponding to ⑥.

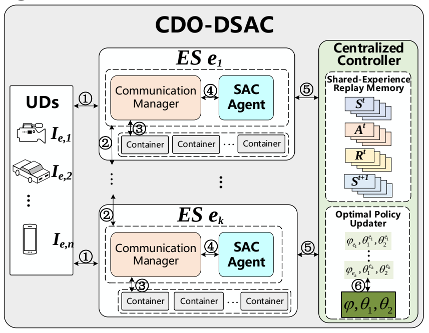


Figure 1. The overview and workflow of CDO-DSAC

CDO-DSAC supports SAC Agents distributed exploration and shared learning experiences. For each SAC Agent, the optimal offloading policy $\pi_{\varphi_{e_i}}^*, \forall e_i \in \mathcal{E}$ can be obtained by maximizing the average return, which is expressed as:

$$\pi_{\varphi_{e_i}}^* = \arg\max_{\pi_{\varphi_{e_i}}} E_{(S^t, A_{e_i}^t) \sim B_{e_i}} [\sum_{t=1}^{\infty} \lambda^t (R_{e_i}^t + \alpha_{e_i} H(\pi_{\varphi_{e_i}}(A_{e_i}^t|S^t)))], \forall e_i \in \mathcal{E}, \quad (3)$$

where φ_{e_i} is the policy parameter of the SAC Agent deployed on e_i, B_{e_i} stores a batch of shared learning experiences randomly selected from D, which improves the diversity of learning sample and reduces the exploration cost of each SAC Agent interacting with the environment.

To speed up the CDO-DSAC training, we set the optimal policy cycle ω to ensure that each SAC Agent can learn the optimal offloading policy. Each iteration has an optimal policy update cycle, and a SAC Agent with the largest average return is selected as the globally optimal policy, and the policy parameters are updated through (4).

$$\pi_\varphi^* \leftarrow \omega\{\arg\max_{\varphi_{e_i}} \pi_{\varphi_{e_i}}^*\}, \exists\, e_i \in \mathcal{E}. \quad (4)$$

B. *Network Structure and Update Process of SAC Agents*

The network structure for the SAC Agent of each ES is shown in Figure 2, where a SAC Agent is taken as an example, with $\forall e_i \in \mathcal{E}$. SAC Agent mainly consists of Actor, Critic, and Experience-Cache. Actor is responsible for interacting with the environment and determining the offloading action for each partition task according to the system state. Critic is responsible for evaluating the offloading policy learned by the Actor. Experience-Cache consists of Replay Memory, Mini-Batch, and Parameter Synchronizer. Replay Memory is used to store the historical learning experiences (S^t, A_e^t, R_e^t, S^{t+1}) learned by SAC Agent. When the learning experience reaches a certain amount, it will be uploaded to the Centralized Controller, and each SAC Agent shares the collected learning experience. Mini-Batch is used to store a batch of learning experiences randomly selected from D and is used for policy optimization. Parameter Synchronizer is responsible for synchronizing the latest policy parameters updated by the Optimal Policy Updater to the Actor and Critic so that each SAC Agent can learn the optimal policy.

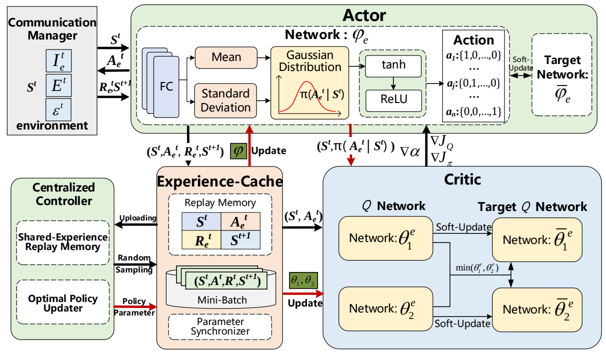


Figure 2. The network structure for the SAC Agent of each ES

(1) Critic. The Critic of each SAC Agent consists of two Q networks and two target Q networks, where double Q networks can overcome the overestimation problem. Q networks take the state-action pair (S^t, $\pi(A_{e_i}^t|S^t)$) under the current offloading policy as input, and output corresponding average return to evaluate the current policy $\pi_{\varphi_{e_i}}$, i.e. Q-value. Although the complete trajectory cannot be obtained during training, a time slot difference is usually used to approximate Q-value, which can be calculated by the following:

$$Q^{\pi_{\varphi_{e_i}}}(S^t, A_{e_i}^t) = R_{e_i}^t + \lambda_{e_i}^t E[Q^{\pi_{\varphi_{e_i}}}(S^{t+1}, A_{e_i}^{t+1})], \forall e_i \in \mathcal{E}. \quad (5)$$

The Q network parameters $\theta_j^{e_i} (j = 1, 2)$ are trained by minimize the Bellm an residual, which is expressed as:

$$J_Q(\theta_j^{e_i})=\frac{1}{2}E_{(S^t,A^t_{e_i})\sim B_{e_i}}[(Q^{\theta_j^{e_i}}(S^t,A^t_{e_i})\text{-}Q^{\pi_{\varphi_{e_i}}}(S^t,A^t_{e_i}))^2],\ \forall\ e_i\in\mathcal{E}\ ,\ j=1,2.\ (6)$$

(2) Actor. The Actor of each SAC Agent consists of an actor network and a target actor network. We use three fully-connected layers to fit the state information, which can output unbounded offloading actions with Gaussian distribution according to the mean and standard deviation. The activat ion function tanh normalizes the offloading actions, maps them to the (-1, + 1) interval, and the segmented activation function Relu is identified as 0 or 1 (no or yes), the specific process is shown in the Actor in Fig. 2. The parameter φ_{e_i} can be trained by minimizing the expected KL-divergence [14] , expressed as :

$$J_\pi(\varphi_{e_i})=E_{S^t\sim B_{e_i}}[E_{A^t_{e_i}\sim\pi_{\varphi_{e_i}}}[\alpha_{e_i}log\pi_{\varphi_{e_i}}(A^t_{e_i}|S^t)\text{-}Q^{\pi_{\varphi_{e_i}}}(S^t,A^t_{e_i})]],\ \forall\ e_i\in\mathcal{E}.\ (7)$$

(3) Update. Critic and Actor require multistep gradient updates to converge, a stable update target is provided using the target network, and the learning stability is improved by updating the target network through an exponential smoothing:

$$\begin{cases}\bar{\theta}_j^{e_i}\leftarrow\tau\theta_j^{e_i}+(1\text{-}\tau)\bar{\theta}_j^{e_i},\ \forall\ e_i\in\mathcal{E},j=1,2,\tau\ll1\\ \bar{\varphi}_{e_i}\leftarrow\tau\varphi_{e_i}+(1\text{-}\tau)\bar{\varphi}_{e_i},\ \forall\ e_i\in\mathcal{E},\tau\ll1\end{cases},\qquad(8)$$

where, $\bar{\theta}_j^{e_i}$ denotes the parameter of the target Q, $\bar{\varphi}_{e_i}$ is the parameter of the target actor, τ is the smoothing coefficient.

(4) Automatic entropy adjustment. Finally, we added an automatic entropy adjustment mechanism to the SAC Agent network to improve the exploration efficiency of the SAC Agent during policy training. When the offloading policy explores a new space, the optimal offloading policy is still unclear, and the α_{e_i} value is increased to improve the exploration ability of SAC Agent. When a state space is learning and the optimal offloading policy is determined, the value of α_{e_i} should be appropriately reduced. The loss of α_{e_i} is minimized by (9), where H_0 is the constant of the target entropy, and the specific solution steps are given in Algorithm 1.

$$J(\alpha_{e_i})=E_{S^t\sim B_{e_i}}E_{A^t_{e_i}\sim\pi_{\varphi_{e_i}}}[\text{-}\alpha_{e_i}\,log\,\pi_{\varphi_{e_i}}(A^t_{e_i}|S^t)\ \text{-}\alpha_{e_i}H_0],\ \forall\ e_i\in\mathcal{E}.\ (9)$$

V. Experimental Verification

A. *Experimental environment and parameter settings*

In a simulated MEC environment, ESs supported inference task offloading requests generated by UDs in a circular area with a service diameter of 150 m. Considering the heterogeneity of computing resources of hardware devices, 5 ESs with the computing power of 30 FLOPs/Byte and 80 UDs with the computing power of 5 FLOPs/Byte were configured in this experiment. Concurrently, we designed a set of environmental variables as the initial parameters of the experiment. The serving capacity of ESs was 30, the network bandwidth was 6Mbps, the transmission power was 20 dB, and the channel gain was 140.7+36.7 log *d*, to control the variables as a benchmark in the experiment. To simulate the randomness of task arrival, we constrained the system task arrival rate to a lognormal distribution [8], whose mean and variance was initialized to 2.0 and 0.7, respectively.

In the MEC environment that provides intelligent services,

Algorithm 1: Distributed SAC-based Partition task Offloading Algorithm

```
Input: System state $S^t$, number of episodes $\varpi$, number of
       initial exploration, Mini-Batch $B_{e_i}$, Shared-
       Experience Replay Memory $D$, Replay Memory
       $D_{e_i}$, optimal policy update period $\omega$.
Output: $\varphi, \theta_1, \theta_2$, Offloading Actions $A$.
Initialization: $\varphi_{e_1}=...=\varphi_{e_k}, \theta_j^{e1}=...=\theta_j^{ek}, j=1,2.$
1.  while episode is not terminated do
2.    for $i$ = 1,2,…,$k$ in parallel do
3.      while initial exploration is not terminated do
4.          Input $S^t$ into Actor and get $A^t_{e_i}$;
5.          Get reward $R^t_{e_i}$ and next state $S^{t+1}$;
6.          Set $D_{e_i}\leftarrow D_{e_i}\cup\{(S^t, A^t_{e_i}, R^t_{e_i}, S^{t+1})\}$;
7.        end while
8.        Set $D\leftarrow D\cup D_{e_i}$
9.      Sample $B_{e_i}=\{(S^t, A^t_{e_i}, R^t_{e_i}, S^{t+1})\}$ from $D$;
10.     for $i$ = 1,2,…,$k$ in parallel do
11.       Update $\theta_1^{e_i}, \theta_2^{e_i}, \varphi_{e_i}$ based on $B_{e_i}$ via (6), (7);
12.       Soft update $\bar{\theta}_1^{e_i}, \bar{\theta}_2^{e_i}, \bar{\varphi}_{e_i}$ via (8);
13.       Update $\alpha_{e_i}$ via (9);
14.     if $\varpi$ mod $\omega$ = 0 then
15.       Select optimal policy $\pi^*_{\varphi_{e_i}}$;
16.       Update $\varphi, \theta_1, \theta_2$ via the optimal policy $\pi^*_{\varphi_{e_i}}$;
17.       Update $\varphi_{e_i}=\varphi, \theta_1^{e_i}=\theta_1, \theta_2^{e_i}=\theta_2$;
18.     end if
19. end while
```

processing image data is the most common in DNN inference. Therefore, we selected three classic and advanced CNN models as benchmarks of the experiment, namely AlexNet, VGG16, and ResNet50, and partitioned the benchmarks according to the network structure, data volume, and UDs computing power to simulate UDs sending partition-based concurrent DNN inference requests to ESs. We used Pytorch to construct AlexNet, VGG16, and ResNet50, used the Berkeley Deep Drive dataset (BDD 100k) [15] for model training, and then implemented CDO-DSAC in the environment to offload target recognition tasks. The latency threshold of the task was set according to the size and type of DNN benchmarks. We deployed SAC Agents on 5 ESs for distributed learning (i.e., k = 5). Each network in Critic and Actor was composed of an input layer, an output layer, and three fully-connected layers. The number of neurons was set to 256, 512, and 256 respectively. In the experiment, t was used as the time slot to discretize the time. Table I summarizes the main hyperparameter settings in CDO-DSAC.

In order to evaluate the performance of the CDO-DSAC, we selected the following four offloading schemes as baseline comparison schemes:

(1) DDPG [7]: A DRL algorithm based on Deep Deterministic Policy Gradient, which is a commonly used task offloading method in the MEC;

(2) Online RL[10]: A RL-based task offloading algorithm to solve the problem of inference task offloading in MEC.

(3) Greedy: It selects the ESs with the smallest predicted

service latency for offloading, which is the default task offloading strategy of many cluster management systems.

(4) Random: It randomly offloads inference tasks to the ESs side. It is the most primitive and easiest-to-think classic offloading algorithm, and it is also a commonly used comparison object in the field of task offloading [16].

TABLE I. MAIN HYPERPARAMETERS

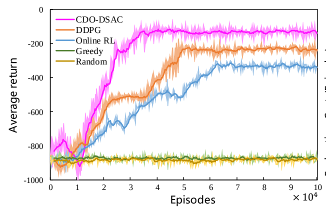
Parameters	Value
Optimal policy update period	10^3
Optimizer	Adam
Learning rate of Actor	10^{-4}
Learning rate of Critic	$3\cdot10^{-4}$
Discount factor	0.99
Temperature coefficient α_{e_i}	0.2
Learning rate of α_{e_i}	10^{-4}
Target smoothing coefficient τ	$5\cdot10^{-3}$
Total number of episodes	10^5

Unless otherwise specified, the hyperparameters involved in the above comparison schemes are consistent with the CDO-DSAC strategy, and each data point in the experimental results is the average of 10 repeated experiments.

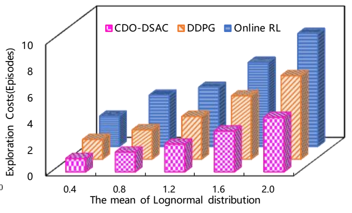
B. *Convergence analysis*

We used 10^5 episodes to train these 5 schemes and compared their convergence. As shown in Figure 3 (a), the solid curve and the shadow area correspond to the mean and standard deviation of the average return of the five schemes, respectively, where the return of CDO-DSAC is the mean of the average return of all SAC Agents. When the episode is 3.96×10^4, CDO-DSAC is close to convergence. Compared with DDPG and Online RL, the convergence speed is increased by 21.1% and 37.5% respectively, and CDO-DSAC can obtain a higher average return. This is because CDO-DSAC based on distributed SAC can learn more experience in less sample space, and SAC Agent based on maximum entropy has stronger exploration ability, and its action selection is more random, to avoid falling into local optimum so that CDO-DSAC can achieve convergence faster and have higher average return. However, the average return of Greedy and Random schemes always hovers around the initial value for they have no learning ability.

Figure 3 (b) shows the exploration cost of CDO-DSAC, DDPG, and Online RL under different task arrival rates. The exploration cost is the number of episodes required to explore when the strategies converge. It can be seen that as the task arrival rate increases, the system state space and the offloading action space also increase, and the exploration cost of the three offloading schemes gradually increases, while the exploration cost of CDO-DSAC is significantly lower than that of DDPG and Online RL. This is because CDO-DSAC supports distributed learning, which can ensure that each SAC Agent can achieve the optimal average return in a cycle, and SAC Agents can share the learning experience obtained through exploration, thus reducing the exploration cost of each Agent, and more sufficient experience data can also help Agents achieve convergence faster. Concurrently, the cumulative discount reward based on maximizing entropy can improve the exploration efficiency of SAC Agents, so that CDO-DSAC has a higher sample learning rate, thus accelerating its training speed and reducing the exploration cost.



(a)

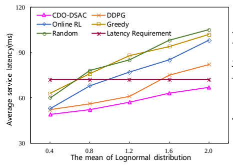


(b)

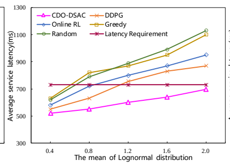
Figure 3. Comparison of average return under different episodes (a) and comparison of exploration costs under different task arrival rates (b).

C. *Accelerating performance evaluation*

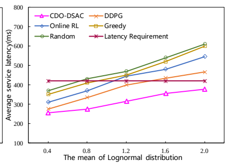
To evaluate the acceleration performance of CDO-DSAC under different task arrival rates and ESs service capacities, we conducted experimental statistics on the average service latency of five offloading schemes on three DNN benchmarks. It can be seen from Figure 4 that compared with the four comparison schemes, CDO-DSAC has the lowest average service latency under different task arrival rates, showing better inference acceleration performance and meeting the latency requirements of benchmarks. However, when the task arrival rate exceeds 1.2, most of the baseline schemes cannot meet the latency requirements. Especially when the task arrival rate is as high as 2.0, CDO-DSAC shows a more obvious acceleration advantage, and its average service latency is reduced by more than 18.3 % and 36.2 % compared with DDPG and Random, respectively. Because the task arrival rate is large at this time, the computing resources of ESs are limited, and there is fierce resource competition among concurrent inference tasks, resulting in the average service latency of the baseline schemes not meeting the requirements. CDO-DSAC based on distributed SAC fully considers the impact of ESs load state and service latency on offloading actions. It can encourage SAC Agents to offload tasks to ESs with lower workloads to obtain higher returns, achieve load balancing among ESs, improve resource utilization, and reduce queuing latency.



(a) AlexNet



(b) VGG16

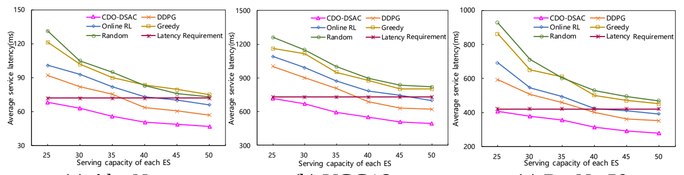


(c) ResNet50

Figure 4. Comparison of service latency under different task arrival rates.

It can be seen from Figure 5 that the average service latency of CDO-DSAC under different ESs service capacities is always the lowest, and is less affected by the change of service capacity, showing better acceleration effect and stability than the comparison schemes. When the service capacity is 40, the average service latency of CDO-DSAC is 19.9% and 38.5% lower than that of DDPG and Random, respectively, because the SAC Agents based on maximum entropy can improve the randomness of the strategy, so that CDO-DSAC can train a higher return and better offloading strategy. DDPG based on deterministic strategy is easy to fall into local optimum. On the other hand, when the service capacity is less than 35, the average service latency of the four comparison schemes is greatly affected by the service capacity, and most of them do not meet the latency requirements of benchmarks, because the service capacity of ESs is extremely limited, resulting in serious resource contention between concurrent inference tasks. CDO-

DSAC can effectively alleviate the resource competition between tasks to reduce queuing latency.

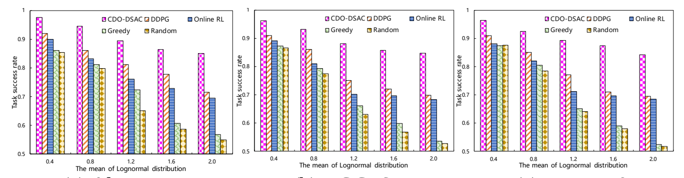


(a) AlexNet (b) VGG16 (c) ResNet50

Figure 5. Comparison of service latency under different service capacities

D. Reliability evaluation

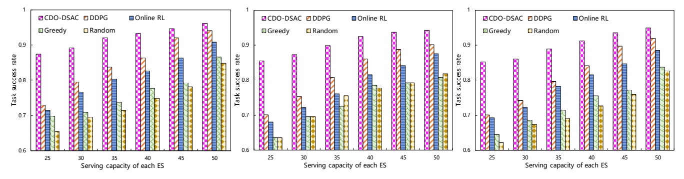
We conducted experimental statistics on the task success rate of three DNN benchmarks under different task arrival rates and serving capacities. It can be seen from Figure 6 that the task success rate of CDO-DSAC under different task arrival rates is higher than that of the other four comparison schemes, showing higher task offloading reliability. Especially when the task arrival rate is as high as 2.0, CDO-DSAC shows more obvious advantages, and its task success rate is more than 18.9% and 22.4 % higher than DDPG and Online RL, respectively. Because the CDO-DSAC strategy based on distributed SAC can fully consider the impact of ESs state and serving latency on offloading actions, on the one hand, it balances the workload between ESs and improves resource utilization; on the other hand, it effectively alleviates the resource competition in the concurrent environment and reduces the queuing delay, so that CDO-DSAC can improve the task success rate of concurrent inference tasks with limited ESs resources.



(a) AlexNet (b) VGG16 (c) ResNet50

Figure 6. Comparison of task success rate under different task arrival rates

It can be seen from Figure 7,the task success rate of CDO-DSAC strategy is higher than 85% under different ESs serving capacities. Compared with the other four baseline comparison methods, CDO-DSAC shows higher task offloading reliability. Especially when the serving capacity is 25, the task success rate of CDO-DSAC is 19.8% and 33.4 % higher than that of DDPG and Random respectively, because the serving capacity of ESs is extremely limited at this time, it is easy to cause task failure due to service overload. CDO-DSAC can balance the load between ESs to reduce resource contention caused by resource constraints and meet the latency requirements. Therefore, it has high task success rate and reliability in extreme environments.



(a) AlexNet (b) VGG16 (c) ResNet50

Figure 7. Comparison of task success rate under different service capacities

VI. Conclusion

In this paper, we model the DNN inference task offloading problem as an MDP with entropy and propose the offloading schedule CDO-DSAC based on distributed SAC to solve the MDP problem. CDO-DSAC is a distributed offloading scheme based on the maximum entropy mechanism. It encourages SAC Agents to optimize in more samples by improving the randomness of exploration, avoiding the policy falling into local optimum. The learning experience can be shared among Agents to better optimize the network, thereby expanding the scale of learning experience data, and reducing the cost of exploration. The experimental results show that CDO-DSAC is superior to the baseline comparison schemes in convergence performance, acceleration performance, stability, and reliability, and has good inference acceleration and load balancing effects. In future work, we will further study how to reduce the total energy consumption of devices in the MEC environment while improving acceleration performance.

References

[1] Xu, D., Li, T., Li, Y. "Edge intelligence: Empowering intelligence to the edge of network." Proceedings of the IEEE 109.11 (2021): 1778-1837.

[2] Gill, S. S., Xu, M., Ottaviani, C., Patros, P., Bahsoon, R., Shaghaghi, A., ... & Uhlig, S. "AI for next generation computing: Emerging trends and future directions. " Internet of Things 19 (2022): 100514.

[3] Wang, X., Han, Y., Leung, V. C., Niyato, D., Yan, X., & Chen, X. "Convergence of edge computing and deep learning: A comprehensive survey."IEEE Communications Surveys & Tutorials 22.2 (2020): 869-904.

[4] Zhang, L., Chen, L., & Xu, J. "Autodidactic Neurosurgeon: Collaborative Deep Inference for Mobile Edge Intelligence via Online Learning." Proceedings of the Web Conference (2021):3111-3123.

[5] Mitsis, G., Tsiropoulou, E. E., & Papavassiliou, S. "Price and risk awareness for data offloading decision-making in edge computing systems." IEEE Systems Journal 16.4 (2022): 6546-6557.

[6] Liu, T., Zhang, Y., Zhu, Y., Tong, W., & Yang, Y. "Online computation offloading and resource Offloading in mobile-edge computing." IEEE Internet of Things Journal 8.8 (2021): 6649-6664.

[7] Wu, W., Yang, P., Zhang, W., Zhou, C. "Accuracy-guaranteed collaborative DNN inference in industrial IoT via deep reinforcement learning." IEEE Transactions on Industrial Informatics (2020): 4988-4998.

[8] Chen, M., & Hao, Y. "Task offloading for mobile edge computing in software defined ultra-dense network." IEEE Journal on Selected Areas in Communications 36.3 (2018): 587-597.

[9] Mohammed, T., Joe-Wong, C., Babbar, R., & Di Francesco, M. "Distributed inference acceleration with adaptive DNN partitioning and offloading." IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE, 2020: 854-863.

[10] Xu, Z., Zhao, L., Liang, W., Rana, O. F. "Energy-aware inference offloading for DNN-driven applications in mobile edge clouds." IEEE Transactions on Parallel and Distributed Systems 32.4 (2020): 799-814.

[11] Tuli, S., Ilager, S. "Dynamic Offloading for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks." IEEE Transactions on Mobile Computing (2020): 940-954.

[12] Ren, D., Gui, X., & Zhang, K. "Adaptive Request Offloading and Service Caching for MEC-Assisted IoT Networks: An Online Learning Approach." IEEE Internetof Things Journal 9.18 (2022): 17372-17386..

[13] Ale, L., King, S. A., Zhang, N. "D3PG: Dirichlet DDPG for Task Partitioning and Offloading With Constrained Hybrid Action Space in Mobile-Edge Computing." IEEE Internet of Things Journal 9.19 (2022).

[14] Christodoulou, P. "Soft actor-critic for discrete action settings." arXiv preprint arXiv:1910.07207 (2019). Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., & Darrell, T.

[15] Yu, F., Xian, W."Bdd100k: A diverse driving video database with scalable annotation tooling."arXiv preprint arXiv: 1805.046 2.5(2018): 6.

[16] Yao, X., Chen, N., Yuan, X. "Performance optimization of serverless edge computing function offloading based on deep reinforcement learning." Future Generation Computer Systems 139 (2023): 74-86.