Adaptive and Collaborative Edge Inference in Task Stream with Latency Constraint

Jinduo Song[1], Zhicheng Liu[1], Xiaofei Wang[1], Chao Qiu[1], Xu Chen[2]
[1]College of Intelligence and Computing, Tianjin University, Tianjin, China
[2] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
Email: {songjinduo, liuzhicheng, xiaofeiwang, chao.qiu}@tju.edu.cn
chenxu35@mail.sysu.edu.cn

***Abstract*—With the rapid development of the Internet of Things (IoT), more and more smart devices are connected to the Internet, implementing Deep Neural Network (DNN) models on edges for collaborative inference via device-edge synergy has become a feasible method for improving application performance in many scenarios. However, when faced with the task stream scenario with latency guarantee such as video surveillance and industrial production line, we need adaptive edge intelligence to make adjustments in real-time according to the changes of the task stream. There are many adaptive edge intelligence technologies in the existing works, such as early-exit mechanism and model selection, but they don't take the requirements of the task stream scenario into consideration. In this paper, we propose a device-edge collaborative inference system based on the early-exit mechanism to solve the problem of adaptive edge intelligence in the task stream scenario. Then, we design an offline dynamic programming (DP) algorithm and an online deep reinforcement learning (DRL) algorithm to dynamically select the exit point and partition point of the branchy model in the task stream, which aims to balance the number of tasks accomplished and task inference accuracy in the system. Experimental results show that the DRL algorithm can achieve performance close to that of the DP algorithm in the task stream scenario.**

***Index Terms*—edge intelligence, task stream, adaptive, DRL**

I. Introduction

There are more and more devices connected to the Internet in the trend of the Internet of Things (IoT). Due to a large amount of device data and the limitation of bandwidth, the traditional way of deploying artificial intelligence (AI) algorithms on cloud servers has been unable to meet the demands of many applications with latency limitations. Deploying models on devices close to data will become a feasible solution to alleviate the bottlenecks of cloud computing.

Many machine learning algorithms, especially Deep Neural Network (DNN), are computing and storage intensive algorithms, and the hardware capabilities of devices currently cannot satisfy the demands of many advanced DNN models, which leads to many difficulties in the process of deploying DNN models on devices [1]–[3]. At the same time, due to the constantly changing environment and different information of different tasks in the task stream scenario, the system should have the capability of adaptive self-regulation. We take the industrial production line as an example. When the production line is busy, more tasks should be accomplished successfully at the expense of performance; when there are fewer tasks in the production line, higher performance should be achieved while accomplishing tasks.

In order to solve the above problems, there has been a lot of works in the recent years. Park et al. [4] propose to train several models with different sizes, the little and fast model can be used to process the input data first, and the big and slow model is only used if the little model cannot achieve the predefined threshold. Taylor et al. [5] propose to train a model selector to choose the best model for different tasks. Although this method of achieving adaptive intelligence through model selection works well, too many models will occupy a large amount of storage space of the devices and make a waste of resources. BranchyNet [6] framework adds several branches to the DNN model structure so that we can choose a branch to exit inference early based on the trade-off between accuracy and latency. Edgent [7] framework combines the early-exit mechanism and model partition technology to maximize the accuracy of model inference while satisfying the latency constraint. This kind of edge intelligence realized through the early-exit mechanism effectively reduces the waste of storage resources by reusing the model structure and is more suitable for deploying on resource-poor devices.

However, there are still some issues that the existing works have not taken into account: 1) **The problem about adaptive and collaborative edge inference cannot be modeled and formulated in detail**. Due to the fact that most works are absolutely based on experiments to implement adaptive and collaborative edge inference, they cannot establish the system model and formulate an optimization problem. 2) **Most works ignore the interaction of adjacent tasks**. Most of the related works do not emphasize the application of the task stream scenario, and the interaction of adjacent tasks can be too obvious to ignore when the application scenario is the task stream.

In this paper, we propose a system model to achieve adaptive edge inference based on the Edgent [7] framework to solve the above problems. In the dynamically changing task stream scenario, our edge inference system continuously selects the exit point and partition point for the task according to the information of the edge queue and current task, which is to improve the task inference accuracy while ensuring that as many tasks as possible can be accomplished within the specified time. In such case, nevertheless, there are new challenges: how to consider the interaction of adjacent tasks in the task stream and how to trade off the number of tasks accomplished and task inference accuracy. If we achieve high accuracy for the current task with longer processing time, there may not be enough time for the next task to process. If low accuracy for the current task is obtained by a conservative strategy, the advantages of adaptive edge intelligence cannot be brought into full play. Therefore, we need to formulate an optimization problem and design decision algorithms to meet the challenges.

In particular, the key contributions of this paper are summarized as follows:

- We establish the system model about adaptive and collaborative edge inference in the dynamic task stream with latency constraint based on the early-exit mechanism and model partition technology, which includes the task model and the computing model. We can further analyze the characteristics of adaptive and collaborative edge inference by establishing a detailed system model.
- We formulate an optimization problem by introducing the interaction of adjacent tasks to the computing model and designing the gain function, which aims to trade off the number of tasks accomplished and task inference accuracy under the latency limitation.
- We propose an offline algorithm using dynamic programming (DP) and an online algorithm using Deep reinforcement learning (DRL) for the optimization problem. Then, we also give a Greedy algorithm and a Random algorithm for comparison in the experiment. Experimental results show that the online algorithm can achieve the performance closer to the offline algorithm than that of other comparative algorithms, and the situation is particularly obvious in the task-intensive stream.

The rest of this paper is organized as follows. In Section II, we introduce the system model which includes the task model and the computing model and formulate an optimization problem. In Section III, we introduce the solution to solve the problem, including an offline algorithm and an online algorithm. We provide simulation results in Section IV and conclude this paper in Section V.

II. System Model

The edge intelligence system we establish is shown in Fig. 1. The system is time-slotted and consists of an edge server and several end devices. The workflow of the system is described as follows: 1) When an end device receives a task, it will send the task information (task data size, latency requirement and etc.) to the edge server and store the task in the task queue. 2) The edge server will make a task offloading decision according to the information of the current task and the task queue of the edge server and

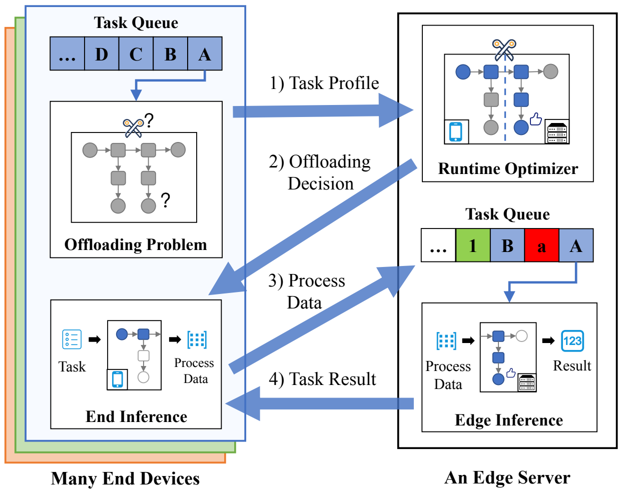


Fig. 1. Adaptive edge inference system

transmit the offloading decision back to the end device. 3) According to the task data and the offloading decision, the end device will process the local part of the task, then transmit the remaining part of the task to the task queue of the edge server. 4) The edge server processes the remaining task from the task queue according to the First Come First Served (FCFS) scheduling principle and transmits the result back to the end device. We assume that the wireless network environment in the system is static and the time of that result returns from the edge server can be ignored [14]. The tasks of the system are generated on end devices according to the Bernoulli process with task arrival rate p, which means there is at most one task generated in the system in each time slot with task arrival rate p [11].

A. *Task model*

We denote the n-th task to be processed in the task stream as $J_n = (D_n, \tau_n, t_n^a)$, where D_n denotes the size of input data, τ_n denotes the maximum tolerable latency for the task, and t_n^a denotes the arrival time of task J_n. We assume that τ_n and D_n are proportional, so we have $\tau_n = k \times D_n, k \in N$. Meanwhile, the structure of the branchy model we use in this paper is shown as Fig. 2, then we denote the number of branches in the branchy model as u, and the number of layers in the i-th model branch as l_i [13]. Next, we denote the exit point and the partition point for the n-th task as m_n and p_n separately, and we have $0 \leq m_n \leq u, 0 \leq p_n \leq l_{m_n}$.

To get the information of DNN's inference time beforehand when making decisions for inference, we need to establish performance prediction models. There are three methods to predict the processing time of DNN in [8] and [9], including Statistical Modeling, Analytical Modeling and Application-specific Profiling. Owing to the difference in hardware and software implementation on different machines, the first two methods have certain errors. The third method requires making different prediction models in advance for different DNN models but with higher reliability, so we choose this method. Then, we establish a regression-

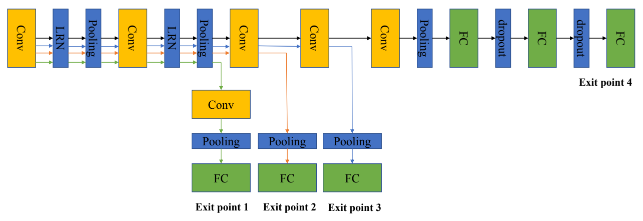


Fig. 2. The structure of the branchy AlexNet model

based prediction model for the processing time and output data size of each layer in the branchy model on the end device and the edge server respectively. Take the former as an example, we use the timeline tool to record the inference time of each layer, and adopt the linear function $y = kx + b$ to fit the inference time of each layer in the model. If $d \in \{0, 1\}$, $f_{i,j}^{d}(D_n)$ denotes the prediction processing time of the j-th layer in the i-th model branch with the input data size D_n, which is on the end device if $d = 0$ or on the edge server if $d = 1$. If $d = 2$, $f_{i,j}^{d}(D_n)$ denotes the prediction output data size of the j-th layer in the i-th model branch with the input data size D_n.

B. Computing model

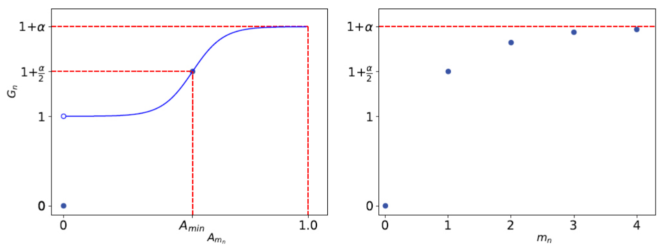
When determining m_n and p_n, we can compute task J_n's processing time on the end device as $t_n^{end} = \sum_{i=1}^{p_n} f_{m_n,i}^{0}(D_n)$, processing time on the edge server as $t_n^{edge} = \sum_{i=p_n+1}^{l_{m_n}} f_{m_n,i}^{1}(D_n)$ and transmission time between the end and the edge as $t_n^{tr} = \frac{f_{m_n,p_n}^{2}(D_n)}{w}$, where w denotes the transmission rate between the end and the edge. Owing to the fact that there is a task queue on each end device and the assumption that there is at most one task generated in the system in each time slot, the adjacent tasks in the task stream may be generated on different end devices, so we assume that task J_h and task J_n are generated adjacently on the same end device, and task J_h is generated first. According to the edge inference system in this paper, we have that the time when the end device starts to process J_n is $t_n^d = max\{t_n^a, t_h^s\}$, the time when the end device starts to transmit J_n is $t_n^s = t_n^d + t_n^{end}$, the time when the edge server starts to process J_n is $t_n^e = max\{t_n^s + t_n^{tr}, t_{n-1}^f\}$, the time when J_n is finished is $t_n^f = t_n^e + t_n^{edge}$, so we can get the total processing time of J_n is $T_n = t_n^f - t_n^a$.

In the branchy model with several exit points, each branch has a specific inference accuracy. Therefore, we denote the accuracy of exit point m_n as A_{m_n}, where $0 \leq A_{m_n} < 1$ and $A_{m_n} = 0$ if and only if $m_n = 0$. In NestDNN [16], Biyi et al. design the cost function that trades off the inference accuracy and the processing time by adding a weight parameter so that a descendant model that meets the requirements can be selected on resource-constrained devices. Similarly, with the aim to improve the accuracy of the task as much as possible while accomplishing as many tasks as possible, we define the gain function of the n-th task as

$$G_n = \lceil A_{m_n} \rceil + \frac{\alpha}{1 + e^{-\beta(A_{m_n} - A_{min})}}, \tag{1}$$

where A_{min} is the minimum inference accuracy of the branchy model, α denotes the performance improvement factor which adjusts the importance of task inference performance in the gain function, and β is a hyper-parameter to tune the impact of changes in inference accuracy on the gain function.

Essentially, the first term in the gain function is used to count the number of tasks accomplished in the task stream by rounding up the inference accuracy, and the second term is the sigmoid function related to the inference accuracy, which is used to express the user's satisfaction with the model performance. Fig. 3 explains how the inference accuracy and the exit points affect the gain function based on the branchy AlexNet model we establish, where $1 + \alpha$ denotes the upper limit of the gain function that the system can obtain from processing a single task and the inference accuracy we can obtain in the system is not lower than A_{min}.



(a) Gain function versus accuracy of the n-th task (b) Gain function versus the exit point of the n-th task

Fig. 3. Explanation diagrams of the gain function based on the branchy AlexNet model

Then, we formulate the optimization problem as follows for the task stream with M tasks in the system.

$$\begin{aligned} &\max_{m_n,p_n} \sum_{n=1}^{M} G_n \\ s.t. \quad & C1: t_1^d = t_1^a, \\ & C2: t_n^f \leq t_n^a + \tau_n, n = 1, 2, ..., M, \\ & C3: 0 \leq m_n \leq u_n, n = 1, 2, ..., M, \\ & C4: 0 \leq p_n \leq l_{m_n}, n = 1, 2, ..., M. \end{aligned} \tag{2}$$

In (2), $C1$ initializes the information of the first task, $C2$ is the latency constraint for the task, $C3$ and $C4$ limit the range of the exit point and the partition point. We can convert this problem into the knapsack problem, M items are needed to pack into the backpack, the volume and value of the item n is T_n and G_n respectively, and the total volume of backpack is $t_M^a + \tau_M$. Therefore, this problem is NP hard.

III. Problem Solution

A. Offline Algorithm

For the problem mentioned in Section II, we can use the DP algorithm to solve it, but the limitation of this method

Algorithm 1: Offline Algorithm

```
Input:
    Number of tasks $M$;
    Arriving time of tasks $t_i^a$;
    Input data volume $D_i$;
    Latency limitation $\tau_i$
Output:
    Maximum gain $G(M, t_M^a + \tau_M)$
1: Set $G(0,t) = 0 (1 \le t \le t_M^a + \tau_M)$
2: for $i = 1$ to $M$ do
3:     if $t_{i-1}^a + \tau_{i-1} < t_i^a$ then
4:         $G(i-1, t_i^a) = G(i-1, t_{i-1}^a + \tau_{i-1})$
5:     end if
6:     for $j = t_i^a$ to $t_i^a + \tau_i$ do
7:         $G(i,j) = \max_{t_i^a \le t < j} \{G(i-1,j), G(i-1,t) + \lceil F(D_i, j-t) \rceil + \frac{\alpha}{1+e^{-\beta(F(D_i,j-t)-Acc_{min})}}\}$
8:     end for
9: end for
```

is that all information of the entire task stream needs to be known beforehand, including data size D_n, arrival time t_n^a, latency limitation τ_n and the number of tasks M. Therefore, this algorithm can only be deployed and processed offline. We use the $F(D,t)$ function to denote the highest accuracy of the exit points that can be selected with data size D under the latency limitation t. Meanwhile, we use $G(i,j)$ to denote the maximum gain that can be obtained when the i tasks are processed under the limitation of j time slots. The offline algorithm is shown in Algorithm 1. According to Algorithm 1, we can get the time complexity of the offline algorithm as $O(M \times \tau^2)$, where τ denotes the average latency limitation of all tasks. Therefore, it is obvious that the running time of the offline algorithm is easily affected by the number of tasks in the task stream and the task latency limitation.

B. Online Algorithm

In the real environment, the information of the task stream cannot be known beforehand. Therefore, although the offline algorithm can get the optimal solution to our problem, it is of little value in practice. Then, we need to design the online algorithm to make decisions in the real-time system. Although it is difficult to get the information of the task stream in advance, the process of the arrival time of tasks is a Bernoulli process with task arrival rate p, and the data size for all tasks obeys normal distribution. Therefore, we model it as a Markov decision process (MDP) and use the DRL algorithm to solve it.

First of all, we need to define states for MDP. To determine whether a task can be successfully finished, we need to know its data size, latency limitation and the total processing time, where the total processing time of the task includes the task waiting time and the processing time on devices. The latter can be easily computed by the data size and regression prediction models, so the information reflected in the state should be data size, latency limitation and task waiting time. According to our system mentioned in Section II, there are two factors that affect the task waiting time of task J_n, which are the time when the end device starts to transmit the previous task on the end device t_h^s and the time when the previous task in the task stream is finished t_{n-1}^f. Therefore, we define the state as $state_n = (D_n, \tau_n - max\{t_h^s - t_n^a, 0\}, max\{t_{n-1}^f - t_n^a, 0\})$, where $\tau_n - max\{t_h^s - t_n^a, 0\}$ means the remaining latency limitation for the current task starts to be processed in the impact of the previous task on the end device, and $max\{t_{n-1}^f - t_n^a, 0\}$ means the overlap processing time of the adjacent tasks in the task stream.

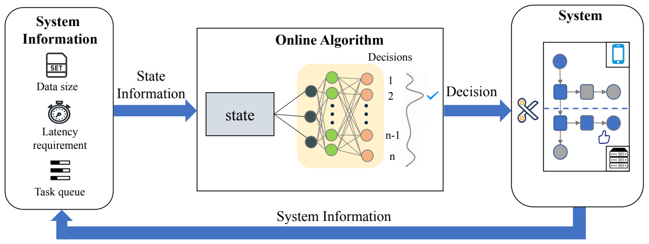


Fig. 4. The workflow of the online algorithm for making decisions

Then, we define the action of MDP. The action selects the exit point and the partition point of the model for each task, which is related to the performance of the system, so we have $action_n = (p_n, m_n)$. Then, to maximize the gain of the system, we directly define the reward as G_n so that we can improve the gain by improving the reward. After establishing the MDP model for the problem, we adopt a DRL algorithm named Double Deep Q Network (DDQN) [10] as our online algorithm, and Fig. 4 shows the workflow of our online algorithm for making decisions. The details of the online algorithm are shown in Algorithm 2.

IV. Simulation Result

A. Experiment Setup

We use virtual machines to simulate several end devices and an edge server, then simulate devices with different computing capabilities by controlling the number of CPUs and operating peaks of virtual machines. The dataset used in this experiment is CIFAR-10 with 10 categories, and each image in CIFAR-10 has 32×32 pixels. The dataset is mainly used in our experiment for model training, model evaluating and simulating the task in the task stream of our system.

The DNN model used in the experiment is an 8-layer AlexNet [12] model, which is mainly used for image classification. The branchy structure we design based on the original AlexNet model is shown in Fig. 2, which has 4 exit points. Although the first branch and the second branch of the branchy AlexNet model have the same number of convolutional layers, the setting of the last convolutional layer is different. If we regard the situation of task failure without running model as an exit point, our branchy model has 5 exit points. Through training, the accuracy of different branches can be 0, 0.527, 0.623, 0.697 and 0.743. Then, we

Algorithm 2: Online Algorithm

```
1: Initialize replay memory $D$ to capacity $N$
2: Initialize action-value function $Q$ with random weights $\theta$
3: Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
4: for $episode = 1$ to $M$ do
5:     Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6:     for $t = 1$ to $T$ do
7:         With probability $\varepsilon$ select a random action $a_t$, otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
8:         Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
9:         Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
10:        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
11:        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
12:        Set
           $y_j = \begin{cases} r_j & \text{if terminal} \phi_{j+1} \\ r_j + \gamma max_{a' \hat{Q}(\phi_{j+1}, a'; \theta^-)} & \text{otherwise} \end{cases}$
13:        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
14:        Every $C$ steps reset $\hat{Q} = Q$
15:    end for
16: end for
```

establish the regression prediction models for the inference latency of each layer and the output data size of each layer according to Section II.

Owing to the hypothesis that we assume the network environment is static, the data transmission rate w in our experimental environment is $1000 kbit/s$. The task generation process of our system is a Bernoulli process with task arrival rate p, and the data size (number of images) of all tasks are normally distributed between [1,10]. The performance improvement factor α is 0.1 and the hyper-parameter β is 16 in the experiment. In the DDQN algorithm mentioned in Section III, the parameter settings we use in the experiment are shown in the Table I.

TABLE I
THE SETTINGS FOR THE DDQN ALGORITHM

Parameters	Values
The number of full connected layers	2
The number of neurons	20
Activation function	tanh
Discount factor	0.9
Replay memory capacity	100000

B. Experimental Result

To show the superiority of our online algorithm in the task stream scenario based on our system, we also design a Greedy algorithm and a Random algorithm for comparison. The Greedy algorithm is to make the optimal offloading decision according to the data size and latency limitation of the current task but does not consider the interaction between adjacent tasks, and this is also the algorithm used by most adaptive edge intelligence at present. The Random algorithm randomly selects the offloading decision without considering any environmental information, so it always achieves the worst performance in our experiment. At the same time, we also compare the offline DP algorithm in the experiment, which has the upper limit of the system performance and can be used as a reference for the performance improvement of the online algorithm. Sum reward in the experiment denotes the sum of the reward of 100 tasks in the task stream.

Fig. 5 shows the sum reward of the system with respect to ratio k, where the number of end devices is 3 and the task arrival rate p is 0.1. As shown in this figure, when the value of k is small, the performance of the DDQN algorithm is higher than that of the Greedy algorithm and closer to that of the DP algorithm. However, with the value of k increasing, the performance of the Greedy algorithm gradually approaches that of the DDQN algorithm. The DDQN algorithm can learn the fixed task arrival rate p of the task stream, so it can achieve better performance than the Greedy algorithm in scenarios with strict task latency constraints. The high value of k indicates that the task of the system can be processed for a longer time, which lessens the interaction between adjacent tasks, so the performance improvement of the DDQN algorithm is often not obvious when k is increasing.

Fig. 6 shows the sum reward of the system with respect to task arrival rate p, where the number of end devices is 3 and the ratio k is 3. The larger the value of p, the more intensive the tasks are in the task stream; the smaller the value of p, the sparser the tasks are in the task stream. As shown in this figure, it can be found that as the task arrival rate p changes, the fluctuation range of the DDQN algorithm performance is the smallest, and the fluctuation range of the Greedy algorithm performance is the largest. When the value of p gradually increases, which means the tasks become more intensive, the performance of the Greedy algorithm declines rapidly, and the decline of the DDQN algorithm is even smaller than that of the DP algorithm, which indicates that the more intensive the task is, the closer the DDQN algorithm is to the optimal performance of the system.

Fig. 7 shows the sum reward of the system with respect to the number of end devices, where the task arrival rate p is 0.1 and the ratio k is 3. As shown in this figure, no matter how the number of end devices changes, the DDQN algorithm is better than the Greedy algorithm, while the DP algorithm has always received the upper limit of the system performance. It is fact that no matter how many end devices are in the system model, the DDQN algorithm can achieve

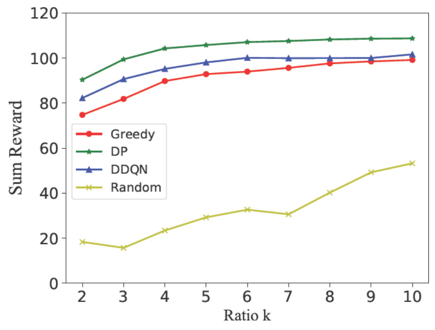


Fig. 5. Sum reward versus ratio k.

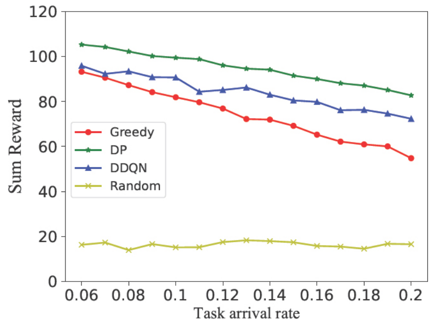


Fig. 6. Sum reward versus task arrival rate p.

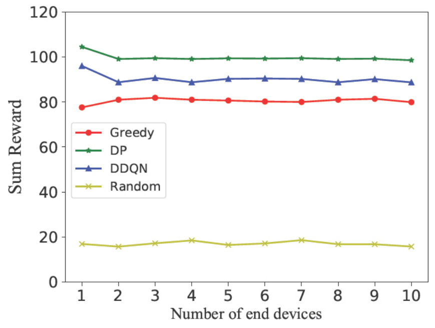


Fig. 7. Sum reward versus the number of end devices.

better performance than that of DP algorithm.

V. Conclusions

In this paper, we establish the system model of the adaptive and collaborative edge inference system based on the early-exit mechanism and model partition technology, which emphasizes the task stream with latency constraint in the system. Then we design the offline DP algorithm and the online DDQN algorithm to trade off the number of tasks accomplished and task inference accuracy. Experimental results show that the performance of the DDQN algorithm is much higher than that of the Greedy algorithm and the Random algorithm in the system with strict latency constraint and intensive tasks. As the latency limitation becomes stricter and the task stream becomes more and more intensive, the performance of the DDQN algorithm gradually approaches the DP algorithm. As far as we know, it is the first work to propose a specific system model and design decision algorithms for the adaptive edge inference system in the task stream with latency constraint, which has great potential in the application of several industrial production lines.

Acknowledgement

This work was supported in part by the National Science Foundation of China under Grant No. 62072332, China NSFC (Youth) through grant No. 62002260; the National Key Research and Development Program of China under Grant No. 2019YFB2101901, No. 2018YFC0809803; the China Postdoctoral Science Foundation under Grant No. 2020M670654; and the Technical Project Funding of State Grid Corporation of China under Contract No. 1400-202055132A-0-0-00.

References

[1] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[2] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[3] Shibo Wang, Shusen Yang, and Cong Zhao. Surveiledge: Real-time video query based on collaborative cloud-edge deep learning. *arXiv preprint arXiv:2001.01043*, 2020.

[4] Eunhyeok Park, Dongyoung Kim, Soobeom Kim, Yong-Deok Kim, Gunhee Kim, Sungroh Yoon, and Sungjoo Yoo. Big/little deep neural network for ultra low power inference. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 124–132. IEEE, 2015.

[5] Ben Taylor, Vicent Sanz Marco, Willy Wolff, Yehia Elkhatib, and Zheng Wang. Adaptive deep learning model selection on embedded systems. *ACM SIGPLAN Notices*, 53(6):31–43, 2018.

[6] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[7] En Li, Zhi Zhou, and Xu Chen. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*, pages 31–36, 2018.

[8] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. Jointdnn: an efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing*, 2019.

[9] Hongshan Li, Chenghao Hu, Jingyan Jiang, Zhi Wang, Yonggang Wen, and Wenwu Zhu. Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 671–678. IEEE, 2018.

[10] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

[11] Xiufeng Huang and Sheng Zhou. Latency guaranteed edge inference via dynamic compression ratio selection. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2020.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[13] Bing Lin, Yinhao Huang, Jianshan Zhang, Junqin Hu, Xing Chen, and Jun Li. Cost-driven off-loading for dnn-based applications over cloud, edge, and end devices. *IEEE Transactions on Industrial Informatics*, 16(8):5456–5466, 2019.

[14] Ji Li, Hui Gao, Tiejun Lv, and Yueming Lu. Deep reinforcement learning based computation offloading and resource allocation for mec. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2018.

[15] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

[16] Fang Biyi, Zeng Xiao and Zhang Mi. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision with edge computing. *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018.