Revised Manuscript (Clean version)

Optimizing CNN Inference Speed over Big Social Data through Efficient Model Parallelism for Sustainable Web of Things

Yuhao Hu[a,1], Xiaolong Xu[a,2,*], Muhammad Bilal[b,3], Weiyi Zhong[c,4], Yuwen Liu[d,5], Huaizhen Kou[e,6] and Lingzhen Kong[e,7]

[a]School of Software, Nanjing University of Information Science and Technology, Nanjing 210044, China
[b]School of Computing and Communications, Lancaster University, United Kingdom
[c]School of Computer Science, Qufu Normal University, China
[d]College of Computer Science and Technology, China University of Petroleum (East China), China
[e]School of Computer Science and Engineering, Nanjing University of Science and Technology, China

ARTICLE INFO

Keywords:
Conv split
resource allocation
inference acceleration
BSD
WoT

ABSTRACT

The rapid development of artificial intelligence and networking technologies has catalyzed the popularity of intelligent services based on deep learning in recent years, which in turn fosters the advancement of Web of Things (WoT). Big social data (BSD) plays an important role during the processing of intelligent services in WoT. However, intelligent BSD services are computationally intensive and require ultra-low latency. End or edge devices with limited computing power cannot realize the extremely low response latency of those services. Distributed inference of deep neural networks (DNNs) on various devices is considered a feasible solution by allocating the computing load of a DNN to several devices. In this work, an efficient model parallelism method that couples convolution layer (Conv) split with resource allocation is proposed. First, given a random computing resource allocation strategy, the Conv split decision is made through a mathematical analysis method to realize the parallel inference of convolutional neural networks (CNNs). Next, Deep Reinforcement Learning is used to get the optimal computing resource allocation strategy to maximize the resource utilization rate and minimize the CNN inference latency. Finally, simulation results show that our approach performs better than the baselines and is applicable for BSD services in WoT with a high workload.

1. Introduction

As a novel implementation pattern of Internet of Things (IoT), Web of Things (WoT) integrates various intelligent devices in web. In traditional IoT, the connection is established only between devices belonging to the same platform for the inconformity of IoT protocols in different platforms [1]. WoT focuses on leveraging the standards of Web to extend the Internet ecosystem to intelligent devices and realizing cross-platform interaction [2, 3]. Through WoT, the potential of IoT is completely inspired and data sharing between arbitrary two devices without limitation is truly realized [4]. In WoT, massive customer data, i.e. big social data (BSD), need to be recorded and analyzed by the intelligent devices for providing corresponding services [5]. Then, the personal information may be stored in multimedia databases for future use.

Thanks to the fast advancement of artificial intelligence and the appearing of BSD, deep learning (DL) has been widely applied in intelligent applications in WoT. Convolutional neural networks (CNNs), regarded as one significant advancement in DL, are widely utilized in computer vision services, such as augmented reality, object classification, object recognition, et al [6, 7, 8]. In spite of the accurate results produced by CNNs, the huge amount of input data which include massive customer information in WoT lead to extremely high computing workload [9], which makes it difficult to infer whole CNN models of intelligent BSD applications on end or edge devices with limited computing resources at a high speed [10, 11, 12]. The traditional solution to this problem is inferring the CNN models on the cloud. Utilizing the robust computational capability of the cloud significantly decreases CNN inference latency, consequently enhancing the quality of service (QoS) for intelligent BSD services in WoT.

However, considering the remote distance between end/edge and cloud as well as the high transmission load caused by the large volume of BSD, the transmission latency is so high in cloud computing. The long transmission time will affect customers' experience, and even cause serious consequences in special occasions like automatic driving. As a result, combining the strong computing power on the cloud with relatively short transmission distance at edge, distributed inference of CNN models in a cloud-edge collaboration framework becomes a breakthrough of the challenge mentioned above. Nevertheless, cloud-edge collaborative inference methods proposed in existing studies either cannot reduce inference latency to the great extent or bring accuracy loss and extral training cost [13, 14, 15]. In addition, with the development WoT, there are more BSD services offloaded from the end [16, 17], which causes higher computing load on the devices to which the services are offloaded [18, 19]. Therefore, for lower inference latency of CNNs with less

*Corresponding author.
202212210036@nuist.edu.cn (Y. Hu); xlxu@nuist.edu.cn (X. Xu); m.bilal@ieee.org (M. Bilal); weiyizhong@qfnu.edu.cn (W. Zhong); B22070001@s.upc.edu.cn (Y. Liu); huaizhenkou@njust.edu.cn (H. Kou); lingzhenkong@njust.edu.cn (L. Kong)

accuracy loss and extral training cost, how to make reasonable convolution layer (Conv) split and computing resource allocation strategies is challenging.

In this paper, we propose a new model parallelism approach which couples Conv split with computing resource allocation to address the issue described above. We set priorities for tasks in WoT and split every Conv into many parts along its height for each CNN. Then, we deploy different parts of each Conv on several edge servers for parallel inference, with a certain number of resources for each part of the corresponding Conv. Compared to former studies, we overcome two challenges. First, the inference latency of CNN still has space for further optimization. Second, the methods for model parallelism proposed before cause large accuracy loss or additional training cost. The main contributions of this paper are as follows:

- We propose a multi-task alternation mechanism considering the huge number of tasks with different priorities in WoT. In the mechanism, the CNN models of intelligent BSD services are inferred alternatively (i.e. anteroposterior Convs of different tasks are inferred at the same time), and services that are assigned greater priority levels are naturally given precedence in processing. This mechanism reduces the time taken for services designated as top priorities to respond as much as possible.
- For each CNN, we design a novel Conv split method which will not cause accuracy loss in theory. This method splits each Conv vertically into multiple parts which are then simultaneously inferred on several separate edge servers, so that it makes full use of available computing resources in all edge servers during the processing of multiple tasks.
- Coupled with the method for Conv split, we design a resource allocation method for inference of each Conv within a specific CNN architecture. We use Deep Reinforcement Learning (DRL) to get the rational resource allocation decision for the minimality of parallel inference latency of each Conv.
- We evaluate the performance of our approach through simulation experiments. We compare the inference latency and accuracy loss of the approach proposed in this paper with those of four baselines. The results prove the applicability of our approach for the services designated as top priorities in WoT with ultra-high workload.

The reminder of this paper is presented as follows. Section 2 shows the related work. Section 3 introduces the system model. Section 4 described the methodology. The results of simulation experiments are presented in 5. Finally, this paper is concluded in Section 6.

2. Related Work

Existing work on model partition is summarized at first in this section. Next, former studies on model parallelism are described. Then, related work on resource allocation is elaborated. At last, we briefly introduce our model parallelism method.

2.1. Model Partition

Model partition means that the DNN is layer-wise partitioned into several parts and deployed on multiple devices to infer sequentially. State-of-the-art work on model partition mainly studies the problems about reduction of inference latency and energy consumption. For inference latency reducing, Hu et al. [13] designed a dynamic adaptive scheme for model partition based on max-min cut algorithm [20]. It supports model partition between edge and cloud under different network conditions in real time. Banitalebi-Dehkordi et al. [21] proposed a method for DNN inference acceleration which joins parameter quantization with model partition. Different from the inference optimization for a single DNN, Duan et al. [22] considered the condition where several DL applications need to be processed in the same time on an end device, and proposed a model partition method joining latency-awareness with task scheduling. This method minimizes the total response time of all tasks on the end device. For energy consumption reducing, Chen et al. [23] solved the problem of DNN model partition and offloading jointly based on particle swarm optimization algorithm and genetic algorithm. Their approach reduces the total energy consumption of all devices.

2.2. Model Parallelism

Model parallelism entails the parallel inference of a DNN model across several devices simultaneously. Note that different from model partition where the whole model is cut into several parts and Convs themselves are not cut, each Conv is cut into several parts in model parallelism. We call the cut of a Conv in model parallelism as "split" in this paper. In terms of model segmentation for distributed inference, the meaning of "partition" and "split" should be distinguished clearly because the two terms represent two different model segmentation dimensions and distributed inference methods. Existing model parallelism methods is divided into two categories, which are spatial split and channel split, respectively. In spatial split, the complete DNN model is copied on each device while the input is split into many parts for parallel inference. Zeng et al. [14] proposed a model parallelism method to optimize inference latency and device energy consumption jointly. In this method, the input is split into several pieces along the height, which are then inferred in parallel on various devices. The optimal input split decision is obtained through linear programming. Zhao et al. [24] proposed a CNN parallel inference framework for edge Internet of Things clusters. They realized the adaptive workload allocation in dynamic scenarios to reduce the latency of CNN inference. In channel split, the different channels of each filter in the Convs are split into several

groups and deployed on different devices for parallel inference. Kim et al. [15] proposed a CNN parallelism approach based on channel split. They grouped the channels of each filter according to the semantics in the input and inferred those grouped channels on several devices in parallel, which enables to reduce the total inference latency a lot.

2.3. Resource Allocation

With the development of edge computing, the design of rational resource allocation decisions for edge devices is of more and more importance. Chen et al. [25] proposed a novel Federated Learning (FL) framework joint with resource allocation. They considered the allocation of computing and communication resources for edge servers which partially merge the raw data before they are uploaded to the cloud. Liang et al. [26] proposed a resource allocation method jointly with task offloading and content caching to improve the efficiency of offloading and caching in mobile edge computing (MEC). They transformed the problem into a convex problem and solved it in a distributed way. Sun et al. [27] considered the cross-server resource allocation in MEC driven by blockchain. They proposed two double auction mechanisms to tackle the problem. Wang et al. [28] proposed a method for edge resource allocation in a dynamic scenario. They formulated the problem of online resource allocation with a nonlinear problem and solved it through the "regularization" technique.

2.4. WoT and BSD

The development of WoT has attracted great attention of researchers. Wang et al. [9] proposed a novel scheme for FL across WoT devices with heterogeneous model architectures, by which training accuracy and speed are improved a lot. Focused on communication security in WoT, Wazid et al. [29] designed an authentication and key agreement scheme based on signature. This scheme largely strengthens security of customers' privacy in interaction between WoT devices. Likewise, as the significant product of WoT, BSD has also been studied a lot. Based on BSD, Nilashi et al. [30] developed a new method for predicting and recommending vegetarian friendly restaurants to customers. In this method, text mining, cluster analysis and predictive learning are jointly used to generate customers' preferences. This BSD analysis-based method improves prediction accuracy and helps customers get better service experience. Feng et al. [31] used BSD to profile e-scooter usages, explore stakeholders, examine operation patterns and do sentiment analysis. This BSD-based study on shared e-scooters helps people deeply understand the emerging shared e-scooter services.

2.5. Introduction of Our Approach

However, the methods for distributed inference of DNN proposed in the former studies are still defective. Model partition cannot realize the acceleration to the greatest extend because its nature of sequential inference. A Conv is inferred in parallel on several servers in model parallelism, but the approaches designed in former studies cause huge accuracy loss and additional training cost. Furthermore, the inference acceleration methods proposed before are seldom joint with resource allocation. The novel model parallelism approach proposed successfully overcome problems mentioned above. It accelerates the processing of BSD based CNN inference tasks to the greatest extend in WoT, without additional training cost and accuracy loss in theory (may still exist accuracy loss in reality, but much lower than other model parallelism methods designed before). Also, we successfully couple the process of Conv split with resource allocation in our model parallelism method.

Table 1
Symbols and Definitions

Symbols	Definitions
N	Number of Convs for a certain CNN
M	Number of edge servers in the scenario
$size_i$	FLOPs of Conv i
x_m^i	The split rate of Conv i on se_m
k_m^i	Number of threads in sub-zone i of a certain resource zone in se_m
η_m	FLOPS of per thread on se_m
K_m	Total number of threads in a certain resource zone of se_m
t_m^i	The transmission time for the partial output of Conv i between co_1 and se_m
t_t^c	The data transmission time from edge to cloud
t_c	The inference time of fully connected layers on c
p_m^i	The height of the part of Conv i inferred on server m without optimization
d_m^i	The height of the part of Conv i inferred on server m after optimization
act_i	Action space of TD3
r_m^i	The resource allocating proportion for sub-zone i of a certain resource zone in se_m
sta_i	State space of TD3
c_m^i	Number of the residual resources in a resource zone of se_m after the allocation for sub-zone i
Z_i	Number of sub-zones to which the resources are not allocated in a certain resource zone
re_i	Return function of TD3

3. System Model

In this section, we present the framework of our approach and establish a mathematical model for it. We aim at reducing the time taken for latency-sensitive intelligent BSD services to respond in WoT, where the number of intelligent service customers is so large and some services with high latency-requirement cannot be responded in a short time. Note that all key symbols in this paper are given in Table 1, which will be used in the following parts of this paper.

3.1. Overview

In our system, we consider a WoT scenario with several edge servers and a large number of intelligent BSD service customers, where workload of edge servers are so large and the fast response requirement of latency-sensitive services cannot meet. In this scenario, we assume that edge

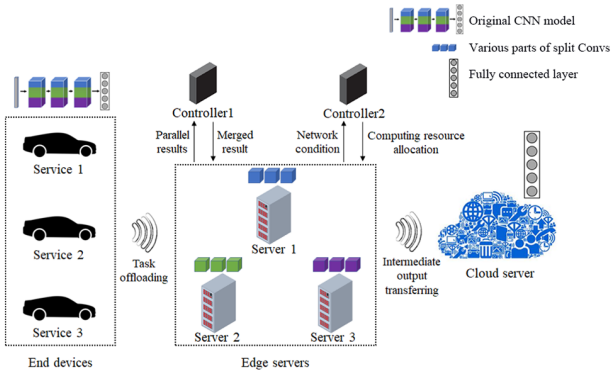


Fig. 1: Overview of our CNN inference acceleration method in WoT.

servers interact with each other through wired connection. Connection between edge and cloud, as well as end and edge, is achieved through wireless network. Each WoT device sends an intelligent BSD service request to controller 1, and service requests are converted to CNN inference tasks in controller 1. As shown in Fig. 1, for the CNN inference task from each customer, each Conv is split into multiple parts and all obtained parts are inferred simultaneously on several separate edge servers. Outputs of each Conv from different servers are merged in controller 1 and then transmitted to the servers for inference of next Conv. The inference of layers with low calculation amount (max pooling layers etc.) are also executed in controller 1. After finishing feature extraction, the activation map is transmitted to fully connected layers on the cloud. Controller 2 makes computing resource allocation decisions for edge servers according to the change of network condition in real time. It is worth emphasizing that the process of Conv split is offline, while the allocation of resources to each Conv is online and dynamically.

In WoT, let $DE = \{de_1, de_2, \cdots, de_Q\}$ denote the set of WoT devices. Q is the number of WoT devices. In terms of the CNN where a service from WoT device is based, the Conv number is denoted as N. Let $SE = \{se_1, se_2, \cdots, se_M\}$ denote the set of edge servers. M is the number of edge servers. Controller 1 and controller 2 are denoted as co_1 and co_2, respectively. The cloud server is denoted as c.

3.2. Multi-task Alternation Architecture

We assume that only few kinds of CNN are used in a certain scenario. Edge servers and controller co_1 have stored the models of these kinds of CNNs beforehand. During the multi-task alternatively processing course, various numbers of computing resources are allocated to different CNN inference tasks. The resource allocation strategy making is divided into two steps. The first step is finished offline. In the first step, controller co_1 creates empty queues for inference tasks of all kinds of CNNs deployed. A queue is corresponding to one of the deployed CNN kinds. In each edge server, the total calculation amount of all kinds of deployed CNNs is computed, and computing resources are allocated to a queue according to the product of total resource number in the edge server and ratio of the corresponding CNN's calculation number to total calculation amount of all deployed CNNs. In terms of resources an edge server allocates to a queue, we regard them belong to a "resource zone". The second step is conducted online. For Convs of each kind of CNN, the corresponding resource zone in each edge server is further partitioned into sub-zones and resources in each sub-zone are allocated to one Conv. The sub-zone partitioning process is context-aware and based on DRL, which is described in detail in Section 4. Note that similar to the relationship between different resource zones, sub-zones in each resource zone are also independent with each other. Computing resources in one sub-zone are particularly allocated to a certain Conv, and other Convs of the corresponding CNN cannot share resources in that sub-zone at any time. The number of sub-zones in a resource zone is equal to the number of Convs of the corresponding CNN. For multiple tasks in succession in a queue, different Convs of the CNN can be inferred with resources in corresponding sub-zones at the same time. Besides, for the same Conv of different tasks in a queue, one Conv of a task cannot begin to be inferred until the inference of this Conv of the previous task is finished.

In the WoT scenario, all tasks converted from services based on the same kind of CNN (TBKCs) are placed within the identical queue, and each task in one queue is given a priority according to its latency-sensitivity. The priority setting is independent among different queues. In each queue, in terms of different task types, the latency-sensitivities of various types of tasks have been recorded and compared in controller co_1 beforehand. For instance, a task converted from one automatic driving service request is more latency-sensitive than that converted from one game running service request, and the automatic driving related task gets a higher priority. In terms of the same task type, the comparison method of different tasks' latency-sensitivities has also been stored in controller co_1 beforehand. For instance, latency-sensitivities of two tasks both converted from automatic driving service requests are compared based on the speeds of vehicles. Besides, latency-sensitivities of two tasks both converted from game running service requests are compared based on the order in which the games are opened. In a queue, for the most latency-sensitive task, such as object recognition on a self-driving car which is moving with a high speed, we set its priority as one, which means that this task should be processed firstly. We use an instance shown in Fig. 2 to analyze the multi-task alternation process. There are three edge servers in WoT and three TBKCs to be processed in the queue. The CNN where the TBKCs are based has three Convs. Note that feature extraction layers excluding Convs are not split and they are inferred integrally in controller co_1 because of their negligible Floating Point Operations (FLOPs). Additionally, the fully connected layers are inferred on the cloud for their large FLOPs and inseparability [32]. The split decisions of Convs have already been made by controller co_2, and Convs deployed on all edge servers have been split according to the obtained decisions

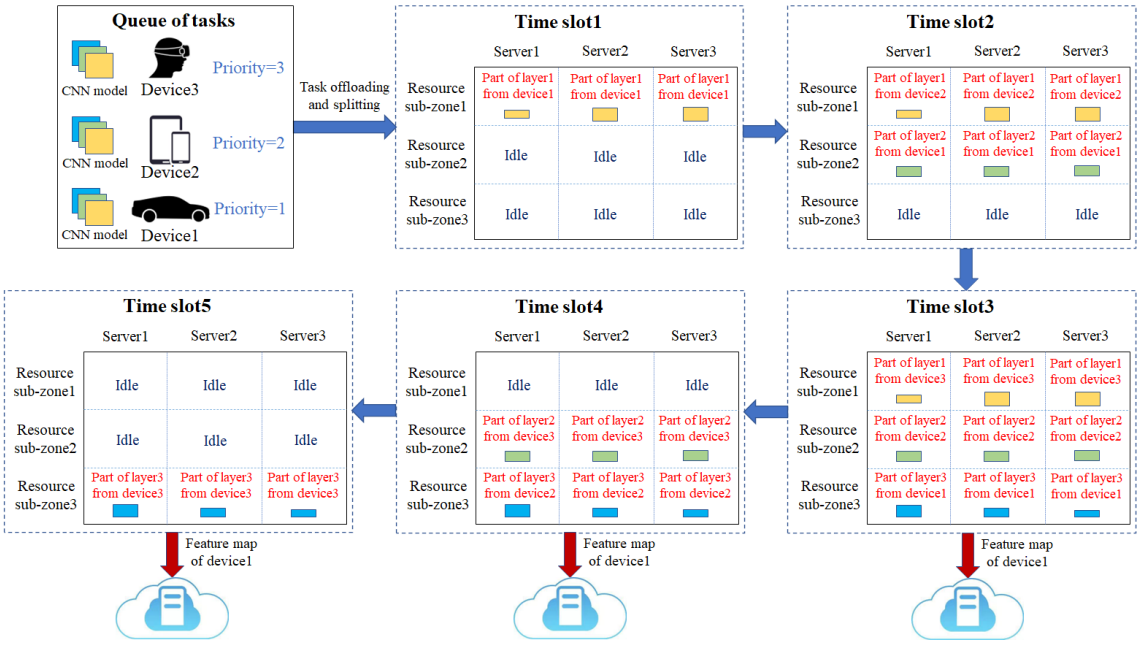


Fig. 2: The architecture for multi-task alternation of three TBKCs.

beforehand. At first, in controller co_1, input of Conv 1 of the task whose priority is 1 (task 1) is split into three parts. Then the three parts are distributed to three distinct edge servers and inferred in parallel with computing resources in sub-zone 1 of each edge server. After finishing the parallel inference of task 1's Conv 1, results from three edge servers are transmitted to controller co_1 and merged. The merged result from controller co_1 is returned to three edge servers and used as input of task 1's Conv 2. The parallel inference process of task 1's Conv 2 is same as that of task 1's Conv 1, while using resources in sub-zone 2 of each edge server. At this moment, considering resources in sub-zone 1 of each edge server are idle, so task 2's Conv 1 begins to be inferred in parallel with sub-zone 1's resources. The parallel inference of task 1's Conv 2 and task 2's Conv 1 is conducted concurrently. After parallel inference of task 1's Conv 2 and task 2's Conv 1 is completed, the results are sent to controller co_1 for merging and the merged results are returned, and then the parallel inference of task 1's Conv 3, task 2's Conv 2 and task 3's Conv 1 starts at the same time. When the parallel inference of task 1's Conv 3, task 2's Conv 2 and task 3's Conv 1 is finished, the merged intermediate result of task 1 is transmitted to the cloud for fully connected layer inference, and parallel inference of task 2's Conv 3 and task 3's Conv 2 begins at edge. The Conv parallel inference continues until the merged intermediate result of task 3's Conv 3 is sent to the cloud. Through the method described above, the response time of services with high priorities is shortened a lot.

3.3. Inference Process of a Single Conv

As shown in Fig. 3, we now analyze the parallel inference process of a single Conv in detail. A Conv is split along its height into several parts, which are allocated to different

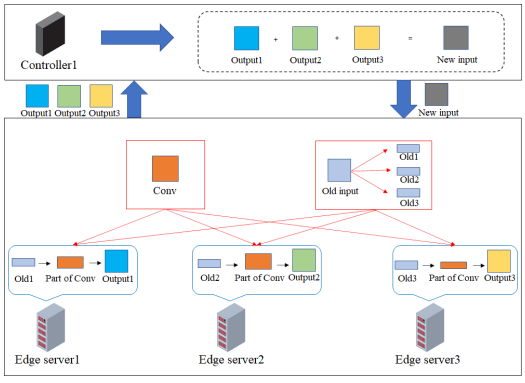


Fig. 3: The architecture for the parallel inference process of a single Conv.

servers for parallel inference. It is well known that in convolution operation, Convs require the surrounding data of a specific region in input for actual computation, and each part of an original Conv does not convolve with each pixel of input. So in order to reduce communication link load and avoid inference accuracy loss caused by input data missing, input of each Conv is also split according to the split decision of the Conv. That is to say, during the parallel inference of each Conv, each part of the Conv gets all needed input data without any data loss and redundancy.

Due to the split of Conv, its input should also be split. We denote the height of input and a Conv as h and b, respectively. On an edge server, it is assumed that the Conv is split from e to l along its height, and the obtained part of the original Conv after splitting is inferred on this edge server. According to the Conv split decision on this edge server, the

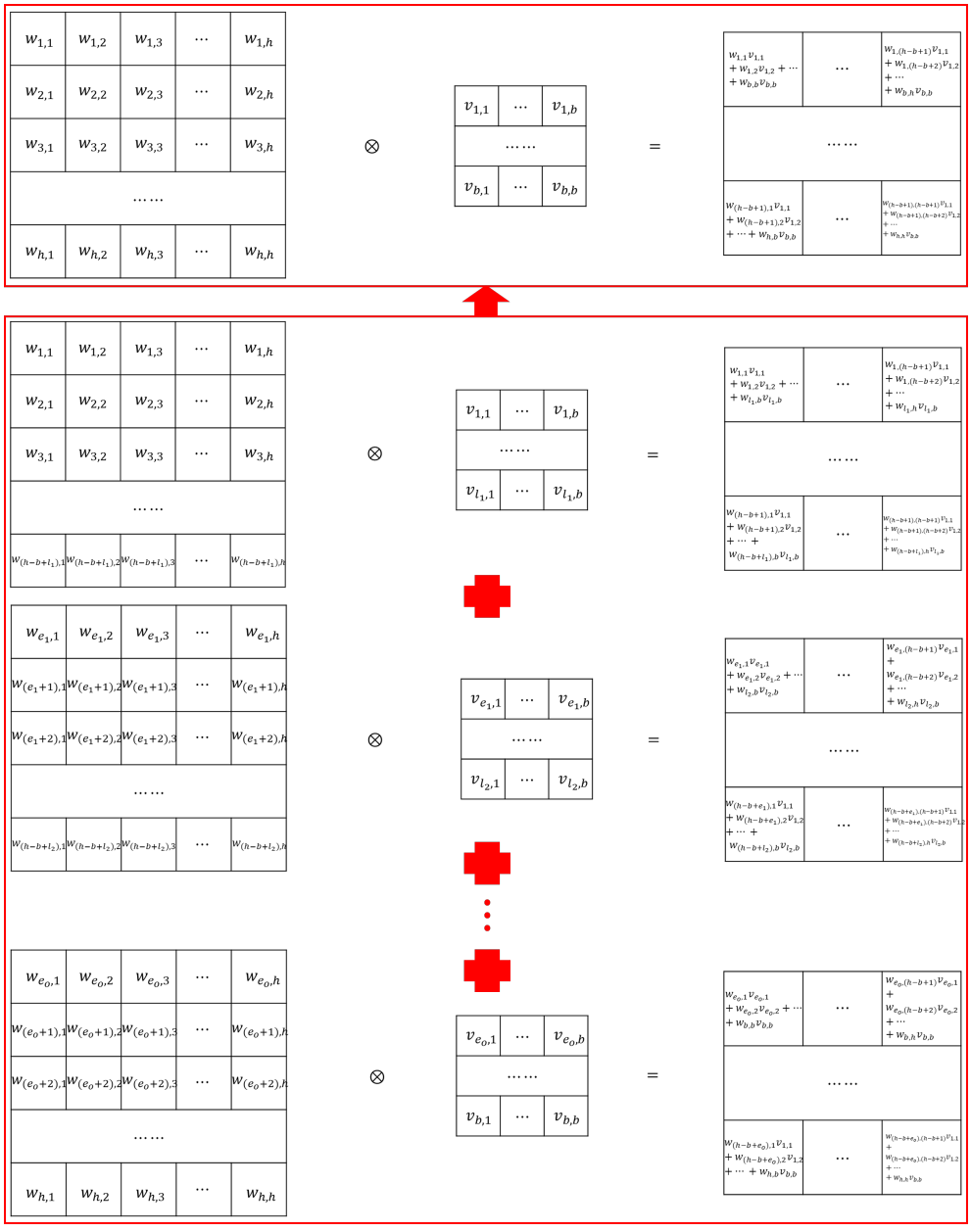


Fig. 4: Proof of non-accuracy loss in theory for our approach.

input is split from s_1 to s_2, which are calculated as $s_1 = e$ and $s_2 = h - (b - l)$.

For each Conv, the size of output from each server is same as that of the result after merging process in controller co_1, which only includes add operations. As shown in Fig. 4, our model parallelism method will not cause accuracy loss in theory.

3.4. Inference Latency Model

CNN inference latency includes computing time and transmission time. The computing time is modeled at first. During parallel inference of Conv i at edge, computing time $t^c_{i,m}$ on edge server se_m is given by

$$t^c_{i,m} = \frac{x^i_m \cdot size_i}{k^i_m \cdot \eta_m}, \tag{1}$$

where x^i_m is Conv i's split ratio on edge server se_m, $size_i$ is FLOPs of Conv i, k^i_m is the number of threads in sub-zone i of a certain resource zone in edge server se_m, and η_m is computing rate (floating point of per second, FLOPS) of per thread in edge server se_m. Computing time t^c_c of fully connected layers on cloud c is given by

$$t^c_c = \frac{size_{full}}{k_c}, \tag{2}$$

where $size_{full}$ is total FLOPs of all fully connected layers of the CNN, k_c is the FLOPS of cloud c.

In terms of data transmission, round-trip transmission time $t^t_{i,m}$ of the partial output of Conv i between controller co_1 and server se_m is given by

$$t^t_{i,m} = \frac{o^{out}_{i,m} + o^{merged}_{i,m}}{ra_m}, \tag{3}$$

where $o^{out}_{i,m}$ and $o^{merged}_{i,m}$ are the sizes of Conv i's partial output from server se_m and merged result of Conv i from controller co_1, respectively. ra_m is data transmission rate between server se_m and controller co_1. Data transmission time t^t_c from edge to cloud c is given by

$$t^t_c = \frac{o^{out}_c}{ra_c}, \tag{4}$$

where o^{out}_c is the size of intermediate result sent to the cloud and ra_c is the edge-cloud transmission rate.

3.5. Problem Formulation

As mentioned above, we reduce the time taken for latency-sensitive intelligent BSD services to respond in WoT with so many customers through a multi-task alternation mechanism, in which the processing of each CNN inference task is independent. In this subsection, we assume that there is only one customer in the scene, and establish a mathematical model to minimize the latency of the single task.

We ignore the end-edge interaction latency and the returning latency of final results from the cloud. The FLOPs of other feature extraction layers are so small that their inference latency can also be neglected. Thus, the problem is formulated as

$$min\ (t_c + t^c_t + \sum_{i=1}^{N} max\{\frac{x^i_m \cdot size_i}{k^i_m \cdot \eta_m} + t^i_m\}) \tag{5}$$

$$s.t.\ m \in [1,\ M], \tag{5a}$$

$$i \in [1,\ N], \tag{5b}$$

$$x^i_m \in [0,\ 1], \tag{5c}$$

$$k^i_m \in \left[1,\ K_m\right], \tag{5d}$$

$$\sum_{m=1}^{M} x^i_m = 1, \tag{5e}$$

$$\sum_{i=1}^{N} k^i_m = K_m, \tag{5f}$$

where x^i_m and k^i_m are the parameters that we need to solve. Constraint (5a) means that the maximal and minimal numbers of servers used to infer each Conv are 1 and M, respectively. Constraint (5b) means that a CNN has 1 Conv at least and N Convs at most. Constraint (5c) means that each split rate of Convs ranges from 0 to 1. Constraint (5d) means that the numbers of resources allocated to each Conv ranges from 1 to K_m. Constraint (5e) means that the summation of all split rates for one Conv is 1. Constraint (5f) means that resources allocated to the Convs within a specific CNN do not exceed the number of resources in the corresponding resource zone, which is K_m.

4. Methodology

In this section, the mathematical model shown above will be solved. This model is a complex min-max problem and hard to be solved in polynomial time. As a result, we make a simplification for this problem. Note that the data transmission time from edge to cloud and the inference latency on the cloud will not be influenced by our model parallelism method, that is to say they are not relative with the solution to the problem, so these parameters can be removed from the original model. The simplified formula is shown below.

$$min\ \sum_{i=1}^{N} max\{\frac{x^i_m \cdot size_i}{k^i_m \cdot \eta_m} + t^i_m\} \tag{6}$$

$$s.t.\ m \in [1,\ M], \tag{6a}$$

$$i \in [1,\ N], \tag{6b}$$

$$x^i_m \in [0,\ 1], \tag{6c}$$

$$k^i_m \in \left[1,\ K_m\right], \tag{6d}$$

$$\sum_{m=1}^{M} x^i_m = 1, \tag{6e}$$

$$\sum_{i=1}^{N} k^i_m = K_m. \tag{6f}$$

As shown in Fig. 5, this problem is solved in two steps, which are Conv split and resource allocation. The two problems are coupled with each other.

4.1. Conv Split

As described in the previous section, the whole process of inference of one Conv includes the computation on the servers and controller co_1, and the data transmission between them. In addition, there is no relation between inference of different Convs, so we can solve the optimal Conv split decision for each Conv independently. For further simplification of this problem, we now temporarily do not consider the data transmission time between controller co_1 and the servers, and set the number of threads in each sub-zone as the average of several random values. After finishing the split of Convs, computing resources are allocated to them in each server in the next sub-section, considering the transmission time. In addition, note that during the process of parallel inference on edge servers, we want to minimize the latency of the server whose inference time is the longest. As a result, the inference time of each server is the same in the ideal condition. So for

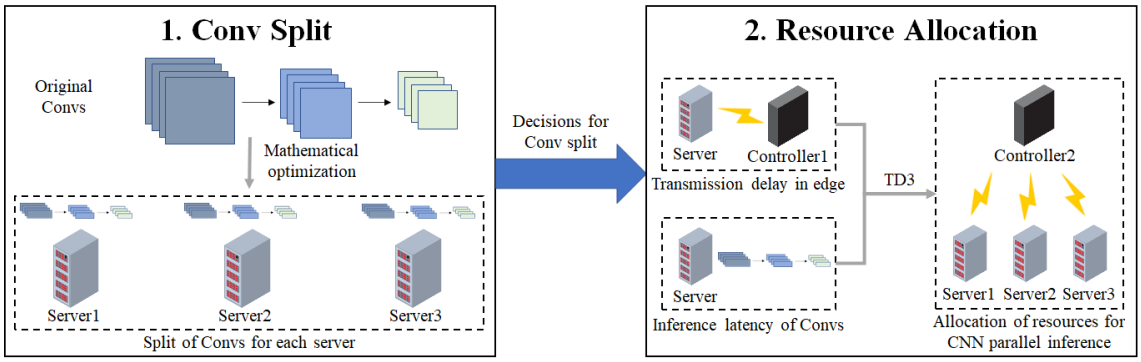


Fig. 5: The workflow of solution to the mathematical model.

the split of each Conv, we get the following formula,

$$\begin{aligned} & x_1^i \cdot a_1^i = x_2^i \cdot a_2^i = \cdots = x_M^i \cdot a_M^i \\ & s.t. \quad \sum_{m=1}^{M} x_m^i = 1, \end{aligned} \tag{7}$$

where $a_j^i = \frac{size_i}{k_j^i \cdot \eta_j}$ is a fixed coefficient. i and j represent the i-th Conv and the j-th server, respectively. We get that $x_j^i = \frac{1}{\left(\sum_{m=1}^{M} \frac{1}{a_m^i}\right) \cdot a_j^i}$.

Proof.

$$\begin{aligned} \sum_{m=1}^{M} \frac{1}{a_m^i} &= \sum \frac{1}{a_1^i} + \frac{1}{a_2^i} + \cdots + \frac{1}{a_M^i} \\ &= \frac{\left(a_2^i a_3^i \cdots a_M^i + a_1^i a_3^i \cdots a_M^i + a_2^i a_3^i \cdots a_{M-1}^i\right) \cdot x_j^i}{\left(a_1^i a_2^i a_3^i \cdots a_M^i\right) \cdot x_j^i} \\ &= \frac{\left(a_1^i a_2^i \cdots a_{j-1}^i a_{j+1}^i \cdots a_M^i\right) \cdot \left(\sum_{m=1}^{M} x_m^i\right)}{\left(a_1^i a_2^i \cdots a_{j-1}^i a_{j+1}^i \cdots a_M^i\right) \cdot \left(a_j^i x_j^i\right)} \\ &= \frac{1}{a_j^i \cdot x_j^i} \Rightarrow x_j^i = \frac{1}{\left(\sum_{m=1}^{M} \frac{1}{a_m^i}\right) \cdot a_j^i}. \end{aligned}$$

Therefore, for the i-th Conv, the size of the part inferred on the m-th server is denoted as $p_m^i = x_m^i \cdot size_i$. However, p_m^i must be an integer. So it needs further optimization. In detail, we get $d_m^i = \lfloor p_m^i \rfloor$ firstly, and calculate $b - \sum_{m=1}^{M} d_m^i$, which is added to d_1^i. Then, we traversal all $\frac{d_m^i}{k_m^i \cdot \eta_m}$ ($m \in [1, M]$) and find the minimal one, which is denoted as t_{min}^i, and the size of the corresponding part of Conv is d_{min}^i. Finally, we get $d_1^i = d_1^i - 1$ and $d_{min}^i = d_{min}^i + 1$. The last two steps are repeated until the maximal one among $\frac{d_m^i}{k_m^i \cdot \eta_m}$ is not smaller than the original $\frac{d_1^i}{k_1^i \cdot \eta_1}$. Then we repeat the operations mentioned above from d_2^i to d_M^i one by one. At last, we get the best split decision. The overview of this approach is given in Algorithm 1.

Algorithm 1 Split of Convs.

Input: Edge server number M, Conv i's height b_i, The split part's height of Conv i on edge server se_m without optimization p_m^i

Output: The set of the split decision for the i-th Conv on M edge servers S={}

1: **for** $m = 1$ *to* M **do**
2: $d_m^i = \lfloor p_m^i \rfloor$
3: **end for**
4: $d_1^i = d_1^i + (b_i - \sum_{m=1}^{M} d_m^i)$;
5: **for** $m = 1$ *to* M **do**
6: $d = d_m^i$
7: **for** $m = 1$ *to* d **do**
8: $t' = \frac{d_m^i}{k_m^i \cdot \eta_m}$
9: Chose the minimal value t_{min}^i among $\{\frac{d_1^i}{k_1^i \cdot \eta_1}, \frac{d_2^i}{k_2^i \cdot \eta_2}, \cdots, \frac{d_M^i}{k_M^i \cdot \eta_M}\}$
10: $d_m^i = d_m^i - 1, d_{min}^i = d_{min}^i + 1$
11: Update set $\{\frac{d_1^i}{k_1^i \cdot \eta_1}, \frac{d_2^i}{k_2^i \cdot \eta_2}, \cdots, \frac{d_M^i}{k_M^i \cdot \eta_M}\}$
12: Find the maximal value t_{max}^i among $\{\frac{d_1^i}{k_1^i \cdot \eta_1}, \frac{d_2^i}{k_2^i \cdot \eta_2}, \cdots, \frac{d_M^i}{k_M^i \cdot \eta_M}\}$
13: **if** $t_{max}^i \geq t'$ **then**
14: break
15: **end if**
16: **end for**
17: put d_m^i into S
18: **end for**
19: **return** S

Now, let us analyze the time complexity of this algorithm. The minimal split unit of a Conv is defined as a *pie*, and each part of the Conv allocated to a certain server is consisted of several *pies*. The core of this algorithm makes up of three nested loops, which is shown in lines 5-18. The outermost loop means that we need to optimize the p_m^i for each server, so we traversal the M edge servers in this loop.

In the two loops inside, we need to find the minimal inference time among all servers with the constant adjusting of d_m^i for each server, which means that the number of steps in these two loops is $d \cdot M$. As a result, the time complexity of this algorithm is $O(M^2 d)$, we assume that $d = \frac{b}{M}$ here, so the time complexity is $O(Mb)$.

There is one thing we need to emphasize that different from the algorithm above, each Conv can also be split directly at first without the calculation of p_m^i. In this method, we just need to traversal all allocation decisions for all *pie*s in a Conv, and choose the best combination. However, it is easy to get that the time complexity of this method is $O(M^N)$, which is much larger than that of Algorithm 1. It proves the high efficiency of our approach.

4.2. Resource Allocation

After getting the Conv split decision, controller 2 should constantly adjusts the number of threads in each sub-zone, with the goal that the inference time of the CNN is minimized. Considering that the cloud-edge transmission time and the inference latency on the cloud are not influenced by the resource allocation method, so we only need to take the inference time in the edge into account. We plan to use Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm [33] in DRL to reach this goal.

For a resource zone, we allocate resources for each sub-zone, which corresponds to a Conv, in sequence. The action space act_i is defined as:

$$act_i = (r_1^i, r_2^i, \cdots, r_M^i), \tag{8}$$

where $r_m^i \in [0, 1]$ ($m \in [1,\ M]$) is the resource allocating proportion for the i-th sub-zone of a certain resource zone in the m-th server. The state space sta_i is defined as:

$$sta_i = (c_1^i, c_2^i, \cdots, c_M^i,\ Z_i), \tag{9}$$

where $c_m^i \in [0, 1]$ ($m \in [1,\ M]$) is the number of the residual and not distributed resources in this resource zone. Z_i represents the number of sub-zones to which the resources are not allocated. We denote that the total number of resources in this resource zone is R. Note that $c_m^{i+1} = c_m^i - \left\lfloor r_m^i \cdot R \right\rfloor$ and $Z_{i+1} = Z_i - 1$, which means that the next state is decided by the current state and action. It proves that the problem conforms to Markov property and can be solved through DRL. As we aim at minimizing the end-to-end latency in the edge, the reward function is defined as:

$$re_i = -max\{\frac{x_m^i \cdot size_i}{k_m^i \cdot \eta_m} + t_m^i | m \in [1,\ M]\}. \tag{10}$$

The training of TD3 is shown in Algorithm 2. We now describe Algorithm 2 in detail. We first conduct initialization (lines 1-3). The number of episodes for training is set as T (line 4), and there are N states in each episode (line 5). Then, the action exploration is executed (lines 6-10). It is ensured that the total allocated resource number for all sub-zones of a resource zone in a server is R (lines 11-17). Then the replay buffer β is updated (line 18). Delayed Policy Updates is utilized in the training process of TD3 (lines 19-28).

After getting the parameters of the actor network, we save them in controller 2, which makes resource allocation decisions in real time.

Algorithm 2 Training of TD3.

Input: Total number of resources in the resource zone R, Number of sub-zones N, Number of edge servers M
Output: Actor network $\pi_\emptyset$

1: Initialize critic networks Q_{θ_1} and Q_{θ_2}, and actor network $\pi_\emptyset$ with random parameters $\theta_1, \theta_2, \emptyset$
2: Initialize target networks $\theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2, \emptyset' \leftarrow \emptyset$
3: Initialize replay buffer β
4: **for** $episode = 1$ *to* T **do**
5: **for** $i = 1$ *to* N **do**
6: **if** $episode < \tau$ **then**
7: $act_i = \varepsilon$
8: **else**
9: $act_i = \pi_\emptyset(sta_i)$
10: **end if**
11: **for** $m = 1$ *to* M **do**
12: **if** $\left\lfloor r_m^i \cdot R \right\rfloor > (R - N\ + 1)$ **then**
13: $\left\lfloor r_m^i \cdot R \right\rfloor > (R - N\ + 1)$
14: **else**
15: $c_m^i = \left\lfloor r_m^i \cdot R \right\rfloor$
16: **end if**
17: **end for**
18: Update β with transition $(sta_i, act_i, re_i, sta_{i+1})$
19: **if** $episode \geq \tau$ **then**
20: Sample H transitions $(sta_h,\ act_h,\ re_h,\ sta_{h+1})_{h=1,\cdots,H}$ from β
21: $y_h \leftarrow re_h + \gamma \cdot min_{g=1,2} Q_{\theta_g'}(sta_{h+1}, \pi_{\emptyset\prime}(sta_{h+1}))$
22: Update critics $\theta_g \leftarrow argmin_{\theta_g} H^{-1} \cdot \sum (y_h - Q_{\theta_1}(sta_h,\ act_h))^2$
23: **if** $episode\ \%\ v == 0$ **then**
24: Update actor network $\pi_\emptyset$:
25: $\nabla_\emptyset J(\emptyset) = N^{-1} \sum \nabla_{\pi_\emptyset(sta_i)} Q_{\theta_1}(sta_h, \pi_\emptyset(sta_i)) \nabla_\emptyset \pi_\emptyset(sta_i)$
26: Update target networks:
27: $\theta_g' \leftarrow \delta\theta_g + (1 - \delta)\theta_g'$
28: $\emptyset' \leftarrow \delta\emptyset + (1 - \delta)\emptyset'$
29: **end if**
30: **end if**
31: **end for**
32: **end for**
33: **return** $\pi_\emptyset$

5. Performance Evaluation

In this section, we evaluate the performance of the proposed model parallelism approach through simulation experiments. We first introduce the experimental settings. Then, we present the convergence of TD3 algorithm used

Table 2
Details of AlexNet Model

Layer number	Layer type	Size	Stride
1	Convolution	$8\times8\times(3or1)\times96$	1
2	Convolution	$8\times8\times96\times256$	1
3	Convolution	7×7×256×384	1
4	Convolution	6×6×384×384	1
5	Convolution	3×3×384×256	1
6	Fully connected	6400×4096	-
7	Fully connected	4096×4096	-
8	Fully connected	4096×100	-

Table 3
Details of VGG-16 Model

Layer number	Layer type	Size	Stride
1	Convolution	$3\times3\times(3or1)\times64$	1
2	Convolution	$3\times3\times64\times64$	1
3	Convolution	3×3×64×128	1
4	Convolution	3×3×128×128	1
5	Convolution	3×3×128×256	1
6	Convolution	3×3×256×256	1
7	Convolution	3×3×256×256	1
8	Convolution	3×3×256×512	1
9	Convolution	3×3×512×512	1
10	Convolution	3×3×512×512	1
11	Convolution	3×3×512×512	1
12	Convolution	3×3×512×512	1
13	Convolution	3×3×512×512	1
14	Fully connected	100352×4096	-
15	Fully connected	4096×4096	-
16	Fully connected	4096×100	-

in resource allocation. We finally compare the inference latency and inference accuracy loss of the proposed model parallelism approach with four baselines.

5.1. Experimental Settings

5.1.1. Device Configuration and Simulation Environmental Parameters

In terms of the device configuration, we use a laptop with a NVIDIA GeForce RTX3070 GPU and the 12-th Gen Intel (R) Core (TM) i9-12900H CPU to execute our simulation experiment. In terms of the environmental parameters, we assume that the computing rate of per thread in each device is same, which is 80 GFLOPS (clock speed is 2.5GHz and 32FLOPs per cycle). Considering that we mainly focus on the split of Convs coupled with resource allocation in CNN parallelism, we assume that the network state is good in WoT. We set the transmission speed of wired connection at edge as values random between 605 Gb/s and 635 Gb/s [34]. The transmission speed of wireless network between edge and cloud is set as a value random between 105 Mb/s and 125 Mb/s [35]. Furthermore, we assume that there are 8 edge servers in the scenario, and 20 threads are allocated to the designated resource zone in each server.

5.1.2. CNN Models and Datasets

For the evaluation of the optimization for inference latency in our approach, AlexNet and VGG-16 [14] are used as the models of CNN inference tasks. We use relative formulas to calculate the latency of data transmission and layer inference for a single customer at first, and then simulate model parallelism and multi-task alternately processing through the multiprocessing module in Python. In the comparison on accuracy loss, we use AlexNet and four datasets, which are CIFAR-100, CIFAR-10 [36], MINIST [37] and FASHIONMINIST [38]. The four datasets include so much information from customers and accord with the characteristic of BSD. The details of used AlexNet and VGG-16 models are respectively shown in Table 2 and Table 3.

5.1.3. Baselines

In order to assess the performance of our approach on inference latency, two baselines are selected, which are shown below.

- **DADS [12]**. In DADS, CNN is partitioned into two parts sequentially through max-min cut algorithm, and the obtained two parts of original network are respectively deployed on the edge and cloud for distributed inference.
- **No Split**. In No Split, the whole model of CNN is inferred on an edge server, without any partition or split.

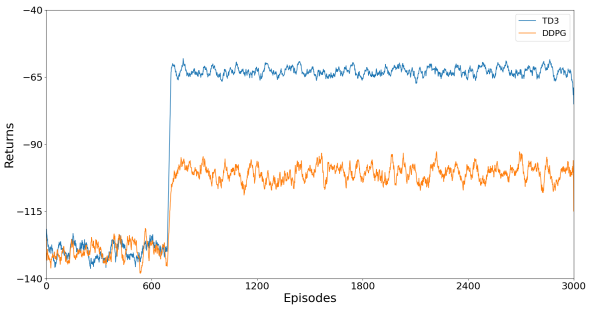
To make the baselines adapt to the WoT scenario with multiple customers, we assume all CNN inference tasks are processed synchronously in the two baselines.

In order to evaluate the performance of our approach on inference accuracy loss reduction, we select two baselines which are shown as follows.

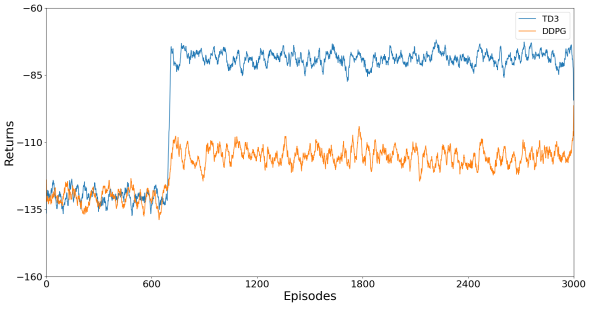
- **CoEdge [14]**. In CoEdge, the complete CNN model is inferred on each edge server while the input is split into many parts dynamically according to the changing of computing and communication resource number at edge.
- **Average Channel Split**. In Average Channel Split, all Convs except the first one of a CNN are split in average and inferred in parallel on the servers.

5.2. Performance of TD3 in Resource Allocation

TD3 is one of improved versions of Deep Deterministic Policy Gradient (DDPG) [39]. Compared to DDPG, improvements on TD3 are mainly reflected in three aspects.

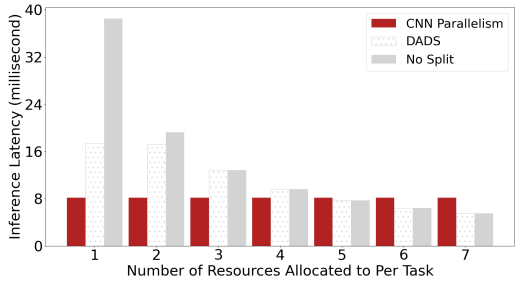


(a) AlexNet

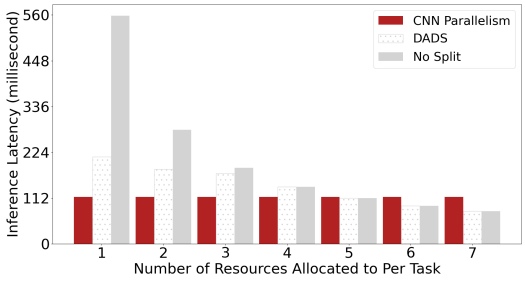


(b) VGG-16

Fig. 6: Comparison between reward functions of TD3 and DDPG.



(a) AlexNet



(b) VGG-16

Fig. 7: Inference latency of a single task with different number of resources.

At first, a new mechanism called Delayed Policy Updates is used in TD3, by which TD3 better balances the frequency of policy updating with the accuracy of the target Q value, making it easier to converge the policy to a better solution. Next, TD3 uses two independent Q networks when computing the target Q value, and selects the smaller Q value as the target, thus suppressing the overestimation of the target Q value and further improving the stability of the training. Finally, TD3 introduces Target Policy Smoothing in action selection. This noise technique helps to improve the exploration ability, making it easier for the agent to explore the unknown action space and obtain a better strategy.

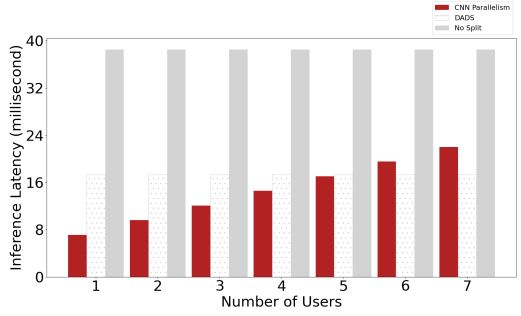
As illustrated in Fig. 6, the convergence speed of TD3 is a little higher than that of DDPG when allocating computing resources not only to AlexNet inference tasks but also to VGG-16 inference tasks. Besides, after converging, the average reward values of TD3 are almost 0.5 times larger than those of DDPG in resource allocation both to AlexNet inference tasks and to VGG-16 inference tasks. These prove the more excellent convergence performance of TD3 compared to DDPG.

5.3. Inference Latency

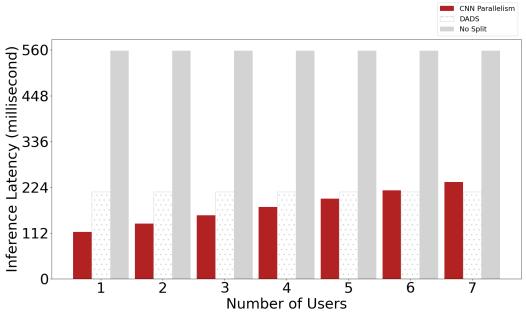
We study the performance of our approach, which is called Model Parallelism here, on inference latency from two angles: the latency of a single task under different levels of customer density, and the total latency of different numbers of tasks with the fixed amount of resources for each task. Note that there may be several task queues in a certain scenario because of the various types of CNNs which are possibly used, and the tasks mentioned in the analysis of our simulation below are in the same queue.

5.3.1. Single Task Latency under Different Customer Densities

In our approach, tasks in each queue are processed one by one and the resources are allocated to each Conv of different types of CNNs, which mean that the number of resources allocated to each task is not influenced by the density of tasks in the scenario. While in the two baselines, in extreme circumstances, each task is just assigned with a single thread. Fig. 7 presents the inference latencies of the three methods under different levels of customer densities, which are reflected by the average threads per task assigned in the baselines. We learn that for one task, the inference latency of our approach does not change under different customer densities, while the latencies of the baselines decrease with the increasing of resources allocated to the task. For AlexNet, in extreme condition, the inference latencies of DADS and No Split are about 17 and 38 milliseconds for a single task, which are around 1 time and 3 times higher than our approach, respectively. When five threads are allocated to the task in the baselines, the inference latencies of DADS and No Split begin to be exceeded by that of our approach. For VGG-16, in extreme condition, the inference latencies of DADS and No Split are about 1 time and 4 times higher than our approach. Like AlexNet, the inference latency of our approach here also exceeds those of DADS and No Split



(a) AlexNet



(b) VGG-16

Fig. 8: Inference latency under different number of customers.

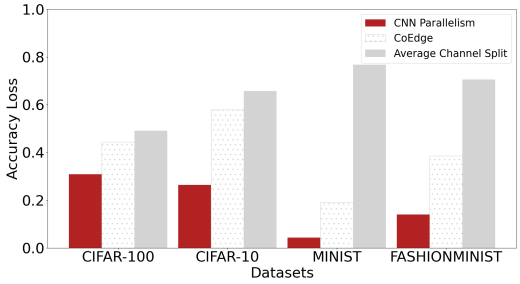


Fig. 9: Accuracy loss using different datasets.

when five threads are allocated to the task in the baselines. Because number of computing resources allocated to each task is not impacted by the workload of edge servers in our approach, the performance of our approach on each task's latency is much more stable than that of the two baselines in WoT where the customer density changes fast. As a result, it is reasonable that when the customer density is small, the performance of the two baselines on each task's latency may exceed that of our approach considering the large number of idle resources in edge servers. The analyses presented above show that the method proposed in this paper is effective when there are many service requests to be dealt with in the scenario and the workload for each edge server is so large.

5.3.2. Total Latency for Multiple Tasks

We now consider multiple tasks to which each is allocated one thread in the baselines. As shown in Fig. 8, because all tasks are processed in parallel in the baselines, the inference latencies of DADS and No Split are fixed regardless of how many tasks we are considering. While due to the multi-task alternation mechanism, the inference latency of our approach nearly goes up linearly as the increase of tasks. In terms of AlexNet, for the first task, the inference latency of our approach is about 12 and 33 milliseconds lower than those of DADS and No Split, respectively. The gaps keep reducing and when the total inference time of six tasks is considered, we find that the latency of our approach exceeds that of DADS. In terms of VGG-16, the condition is similar to AlexNet, and the total inference time of our approach also begins to exceed that of DADS from the 6-*th* task in the queue. In the two baselines, considering that all tasks are processed synchronously, total latency of processing all tasks is equal to the latency of processing a single task. While in our approach, tasks in a queue are processed following the priority order, and total latency of processing all tasks increases with the growth of task number. So it is common that the total latency of our approach may be higher when the task number becomes large. Besides, for the both CNN models, the inference latency of No Split keeps much higher than those of our approach and DADS in the first seven tasks. It generally shows that our approach is available for the several services with the highest priorities in the scenario, where exists a huge amount of service requests waited to be processed.

5.4. Accuracy Loss

In spite of the proof of non-accuracy loss in theory for our approach above, we find that the accuracy loss exists in reality. This is because that during the convolution process, many values in the feature maps are infinite decimals, and the split of Convs breaks the integrity of convolution operations, which produce much errors in the process of result merging. Furthermore, considering that each Conv in a CNN is split, the errors constantly enlarge layer by layer. As shown in Fig. 9, in terms of CIFAR-100, the accuracy loss of our approach is about 30%, while the losses of CoEdge and Average Channel Split are about 44% and 49%, which are both about 0.5 times higher than that of our approach. In terms of CIFAR-10, the accuracy loss of our method is around 26%, while the losses of CoEdge and Average Channel Split are around 58% and 65%, which are both more than twice higher than that of our method. For MINIST, the accuracy loss of our method is about 4%, while the losses of CoEdge and Average Channel Split are about 19% and 77%, which are nearly 5 times and more than 19 times as high as that of our method, respectively. For FASHIONMINIST, the accuracy loss of our method is about 14%, while the losses of CoEdge and Average Channel Split are about 38% and 71%, which are nearly 4 times and more than 5 times as high as that of our method, respectively. In a word, although the accuracy losses in the three methods are various in terms of the four datasets, our approach outperforms CoEdge and Average Channel Split in inference accuracy in general.

6. Conclusion

Generally, the CNN parallelism approach proposed in this paper largely reduces time taken for intelligent BSD services to respond in WoT. This approach includes two coupled problems, which are Conv split and resource allocation. Simulation experiments prove that our approach outperforms the baselines, and is suited to process latency-sensitive intelligent BSD services in WoT where the workload is extremely high. Nevertheless, the CNN inference latency is promising to be further reduced by shortening data transmission time, which is an important research direction of our future work. In addition, the multi-task alternation mechanism in this paper is unfair for the tasks with relatively low priorities, which can be further optimized.

Acknowledgments

This work was supported in part by the Major Research plan of the National Natural Science Foundation of China under Grant (No. 92267104), Natural Science Foundation of Jiangsu Province of China under Grant (No. BK20211284), and Financial and Science Technology Plan Project of Xinjiang Production and Construction Corps under Grant (No. 2020DB005).

References

[1] J. C. Provoost, A. Kamilaris, L. J. Wismans, S. J. Van Der Drift, M. Van Keulen, Predicting parking occupancy via machine learning in the web of things, Internet of Things 12 (2020) 100301.

[2] V. Charpenay, A. Zimmermann, M. Lefrançois, O. Boissier, Hypermedea: A framework for web (of things) agents, in: Companion Proceedings of the Web Conference 2022, 2022, pp. 176–179.

[3] W. Fang, W. Zhang, W. Chen, L. Yi, W. Gao, Pdtm: Poisson distribution-based trust model for web of things, in: Companion Proceedings of the Web Conference 2021, 2021, pp. 85–89.

[4] L. Sciullo, L. Gigli, A. Trotta, M. Di Felice, Wot store: Managing resources and applications on the web of things, Internet of Things 9 (2020) 100164.

[5] N. Misra, Y. Dixit, A. Al-Mallahi, M. S. Bhullar, R. Upadhyay, A. Martynenko, Iot, big data, and artificial intelligence in agriculture and food industry, IEEE Internet of things Journal 9 (9) (2020) 6305–6324.

[6] J. Redmon, A. Farhadi, Yolo9000: better, faster, stronger, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 7263–7271.

[7] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, Advances in neural information processing systems 28 (2015).

[8] X. Hou, Y. Guan, T. Han, N. Zhang, Distredge: Speeding up convolutional neural network inference on distributed edge devices, in: 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2022, pp. 1097–1107.

[9] K. Wang, Q. He, F. Chen, C. Chen, F. Huang, H. Jin, Y. Yang, Flexifed: Personalized federated learning for edge clients with heterogeneous model architectures, in: Proceedings of the ACM Web Conference 2023, 2023, pp. 2979–2990.

[10] S. Chen, Q. Li, M. Zhou, A. Abusorrah, Recent advances in collaborative scheduling of computing tasks in an edge computing paradigm, Sensors 21 (3) (2021) 779.

[11] X. Hou, T. Han, Trustserving: A quality inspection sampling approach for remote dnn services, in: 2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), IEEE, 2020, pp. 1–9.

[12] H. Liang, Q. Sang, C. Hu, D. Cheng, X. Zhou, D. Wang, W. Bao, Y. Wang, Dnn surgery: Accelerating dnn inference on the edge through layer partitioning, IEEE transactions on Cloud Computing (2023).

[13] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, Y. Li, Task partitioning and offloading in dnn-task enabled mobile edge computing networks, IEEE Transactions on Mobile Computing (2021).

[14] L. Zeng, X. Chen, Z. Zhou, L. Yang, J. Zhang, Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices, IEEE/ACM Transactions on Networking 29 (2) (2020) 595–608.

[15] J. Kim, Y. Park, G. Kim, S. J. Hwang, Splitnet: Learning to semantically split deep networks for parameter reduction and model parallelization, in: International Conference on Machine Learning, PMLR, 2017, pp. 1866–1874.

[16] R. Li, Z. Zhao, X. Zhou, G. Ding, Y. Chen, Z. Wang, H. Zhang, Intelligent 5g: When cellular networks meet artificial intelligence, IEEE Wireless communications 24 (5) (2017) 175–183.

[17] J. Wang, J. Hu, G. Min, A. Y. Zomaya, N. Georgalas, Fast adaptive task offloading in edge computing based on meta reinforcement learning, IEEE Transactions on Parallel and Distributed Systems 32 (1) (2020) 242–253.

[18] M. Xue, H. Wu, G. Peng, K. Wolter, Ddpqn: An efficient dnn offloading strategy in local-edge-cloud collaborative environments, IEEE Transactions on Services Computing 15 (2) (2021) 640–655.

[19] L. Tong, Y. Li, W. Gao, A hierarchical edge cloud architecture for mobile computing, in: IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE, 2016, pp. 1–9.

[20] Y. Boykov, V. Kolmogorov, An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision, IEEE transactions on pattern analysis and machine intelligence 26 (9) (2004) 1124–1137.

[21] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, Y. Zhang, Auto-split: a general framework of collaborative edge-cloud ai, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 2543–2553.

[22] Y. Duan, J. Wu, Joint optimization of dnn partition and scheduling for mobile cloud computing, in: 50th International Conference on Parallel Processing, 2021, pp. 1–10.

[23] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, G. Min, Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments, IEEE Transactions on Parallel and Distributed Systems 33 (3) (2021) 683–697.

[24] Z. Zhao, K. M. Barijough, A. Gerstlauer, Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37 (11) (2018) 2348–2359.

[25] S. Luo, X. Chen, Q. Wu, Z. Zhou, S. Yu, Hfel: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning, IEEE Transactions on Wireless Communications 19 (10) (2020) 6535–6548.

[26] C. Wang, C. Liang, F. R. Yu, Q. Chen, L. Tang, Computation offloading and resource allocation in wireless cellular networks with mobile edge computing, IEEE Transactions on Wireless Communications 16 (8) (2017) 4924–4938.

[27] W. Sun, J. Liu, Y. Yue, P. Wang, Joint resource allocation and incentive design for blockchain-based mobile edge computing, IEEE transactions on wireless communications 19 (9) (2020) 6050–6064.

[28] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, L. Tang, Neurosurgeon: Collaborative intelligence between the cloud and mobile edge, ACM SIGARCH Computer Architecture News 45 (1) (2017) 615–629.

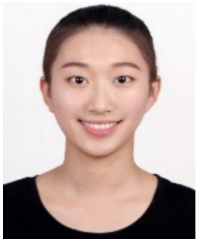
[29] M. Wazid, A. K. Das, K.-K. R. Choo, Y. Park, Scs-wot: Secure communication scheme for web of things deployment, IEEE Internet of Things Journal 9 (13) (2021) 10411–10423.

[30] M. Nilashi, H. Ahmadi, G. Arji, K. O. Alsalem, S. Samad, F. Ghabban, A. O. Alzahrani, A. Ahani, A. A. Alarood, Big social data and customer decision making in vegetarian restaurants: A combined machine learning method, Journal of Retailing and Consumer Services 62 (2021) 102630.
[31] Y. Feng, D. Zhong, P. Sun, W. Zheng, Q. Cao, X. Luo, Z. Lu, Micromobility in smart cities: A closer look at shared dockless e-scooters via big social data. arxiv 2020, arXiv preprint arXiv:2010.15203 5 (2020).
[32] Y. H. Oh, Q. Quan, D. Kim, S. Kim, J. Heo, S. Jung, J. Jang, J. W. Lee, A portable, automatic data qantizer for deep neural networks, in: Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques, 2018, pp. 1–14.
[33] S. Fujimoto, H. Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: International conference on machine learning, PMLR, 2018, pp. 1587–1596.
[34] R. Ullah, S. Ullah, A. Almadhor, H. S. Alwageed, A. A. Al-Atawi, J. Ren, S. Chen, A high-capacity optical metro access network: Efficiently recovering fiber failures with robust switching and centralized optical line terminal, Sensors 24 (4) (2024) 1074.
[35] T. Ji, X. Wan, X. Guan, A. Zhu, F. Ye, Towards optimal application offloading in heterogeneous edge-cloud computing, IEEE Transactions on Computers (2023).
[36] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images (2009).
[37] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[38] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747 (2017).
[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971 (2015).

Yuhao Hu received the B.S. degree in mechanical engineering from Nanjing Tech University in 2022. He is currently pursuing the master's degree in software engineering. His research interests include mobile edge computing, big data, Internet of Things, and deep learning.

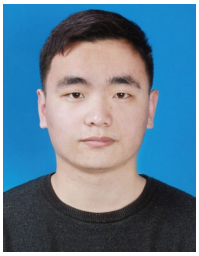
Xiaolong Xu received the Ph.D. degree in computer science and technology from Nanjing University, China, in 2016. He was a Research Scholar with Michigan State University, USA,from April 2017 to May 2018. He is currently a Full Professor with the School of Software, Nanjing University of Information Science and Technology. He has published more than 100 peer-review articles in international journals and conferences, including the IEEE TPDS, IEEE T-ITS, IEEE TII, ACM TOSN, ACM TOMM, ACM TIST, IEEE ICWS, ICSOC, etc. He was selected as the Highly Cited Researcher of Clarivate 2021 and 2022. His research interests include Big data, Internet of Things and Deep learning.

Muhammad Bilal received the Ph.D. degree from the Korea University of Science and Technology. He served as a Postdoctoral Research Fellow at Korea University's Smart Quantum Communication Center. In 2018, he joined Hankuk University of Foreign Studies, South Korea, where he was as an Associate Professor with the Division of Computer and Electronic Systems Engineering. In 2023, he joined Lancaster University, where he is working as a Senior Lecturer (Associate Professor) with the School of Computing and Communications. With over 14 years of experience, Dr. Bilal has actively participated in numerous research projects funded by prestigious organizations like IITP, MOTIE, NRF, and NSFC, etc. He is the author/co-author of 140+ articles published in renowned journals and holds multiple US and Korean patents. His research interests include Network Optimization, Cyber Security, the Internet of Things, Vehicular Networks, Information-Centric Networking, Digital Twin, Artificial Intelligence, and Cloud/Fog Computing. Dr. Bilal actively contributes to the academic community as an editorial board member for esteemed journals, including IEEE Transactions on Intelligent Transportation Systems, IEEE Internet of Things Journal, Alexandria Engineering Journal, and more. He has also served on the Technical Program Committees of prestigious conferences like IEEE VTC, IEEE ICC, ACM SigCom, and IEEE CCNC.



Weiyi Zhong received the BEng degree from Qufu Normal University, China in 2018. She is currently a master student at Qufu Normal University, China. Her research interests include recommender system and service computing.

Yuwen Liu (Graduate Student Member, IEEE) received the master's degree from the School of Computer Science, Qufu Normal University, Rizhao, China, in 2022. She is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, China University of Petroleum, Qingdao, China. Her research interests include recommender systems and privacy protection.



Huaizhen Kou is currently a Ph.D. candidate at Nanjing University of Science and Technology. He has received his B.S. and M.S. degrees in Computer Science from Qufu Normal University in 2018 and 2020, respectively. His research interests include Service Computing, Recommender Systems, and Link Prediction.

Lingzhen Kong received her Master degree in the School of Computer Science, Qufu Normal University, Rizhao, China, in 2019. She is currently a Ph.D. candidate in the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. Her research interests are healthy big data and privacy protection.