Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

A cloud-edge collaborative task scheduling method based on model segmentation

Chuanfu Zhang[1,2], Jing Chen[1,2*], Wen Li[1,2], Hao Sun[1,2], Yudong Geng[1,2], Tianxiang Zhang[1,2], Mingchao Ji[1,2] and Tonglin Fu[1,2]

Abstract

With the continuous development and combined application of cloud computing and artificial intelligence, some new methods have emerged to reduce task execution time for training neural network models in a cloud-edge collaborative environment. The most attractive method is neural network model segmentation. However, many factors affect the segmentation point, such as resource allocation, system energy consumption, load balancing, and network Bandwidth allocation. Some segmentation methods consider the shortest task execution time, which ignores the utilization of resources at the edge and can result in resource waste. Additionally, these factors are difficult to measure, which presents a challenge in calculating the best segmentation point to achieve the goal of maximum resource utilization and minimum task execution time. To solve this problem, this paper proposes a cloud-edge collaborative task scheduling method based on model segmentation (CECMS). This method first analyzes the factors affecting the segmentation point of the model and then obtains accurate factors that affect the segmentation point calculation through the pre-execution method. Furthermore, a multi-objective solution algorithm is improved to calculate the optimal model segmentation point. And tasks are separately offloaded to the edge and cloud based on the optimal model segmentation point. Finally, the experiments are conducted to verify the effectiveness of this method. Finally, the effectiveness of the CECMS method was verified through simulation experiments. Compared with the Dynamic Adaptive DNN Surgery (DADS) method and an adaptive DNN inference acceleration framework algorithm with end–edge–cloud collaborative computing algorithm (ADC), CECMS achieves the same effectiveness as DADS and ADC in optimizing task execution time by comprehensively considering the utilization of edge resources and minimizing task execution time, while also effectively ensuring resource utilization.

Keywords Cloud-edge collaboration, Model segmentation, Task execution time, Resource utilization

Introduction

Cloud computing resource scheduling and IoT security have made great progress through continuous development [1]. Many studies focus on cloud computing scheduling, including algorithms based on deep reinforcement learning [2], system security [3], ordinal optimization [4], and multi-objective trust-awareness [5]. The time cost of AI tasks can be reduced by applying cloud computing in AI [6]. The conventional approach to combining cloud computing with AI is to upload all AI tasks and data to be computed to the cloud to leverage its high computing capacity, which, however, potentially results in significant transmission delay, hinders task completion, and incurs high energy cost in the case of a long distance between the cloud and the user and a large amount of calculation data. The massive transmission delay associated

*Correspondence:
Jing Chen
jingchen94@163.com
[1] Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan, China
[2] Shandong Provincial Key Laboratory of Computer Networks, Shandong Fundamental Research Center for Computer Science, Jinan, China

with traditional cloud computing limits its effectiveness in real-time medical treatment [7, 8], despite its applications in the medical field. Similarly, combining traditional cloud computing with artificial intelligence proves ineffective in handling latency-sensitive tasks such as autonomous driving [9] and networked vehicles [10]. The emergence of edge computing has addressed significant transmission delays brought by the distance between the cloud and the end user [11]. Edge computing, which refers to adding a computing center near the end user [12], decentralizes some of the cloud's capabilities to the edge to offload the task uploaded by the end user to the cloud due to insufficient local computing resources. Edge computing has led to the development of a more advantageous cloud-edge collaborative architecture. If edge node has computing resources far from being sufficient to complete the task offloaded by the end-user, it will further offload the task to the cloud [13], which avoids the extensive transmission delay caused by uploading tasks from some end users far from the cloud and enables real-time interaction. Combining the advantages of cloud computing and edge computing, cloud-edge collaborative computing makes it more efficient to process massive computing tasks, quicker to execute real-time tasks and more effective to handle delay-sensitive tasks [14].

The training task of an artificial intelligence neural network model is computationally intensive and requires large amounts of data. When the neural network model training task is offloaded into a cloud-edge collaborative system, the edge computing resource may not be able to complete the computation of the entire task [15]. Therefore, tasks are often uploaded to the cloud for unloading, which can result in significant transmission delays caused by the massive data required for training, thus requiring more time for completing the tasks. To make better use of the cloud-edge collaboration system, the concept of splitting the neural network model first prior to computing is proposed. The DNN model is partitioned and executed in a distributed manner by adjusting the DNN partition points to achieve the optimal latency or mobility energy [16, 17].

In summary, Segmenting neural network models can significantly reduces the time required for completing neural network model tasks, which, however, needs the corresponding segmentation points of the model to be accurately calculated. Incorrect point calculation may result in increased task completion time, resource waste, deadlock, and even failure in task offloading. There are a number of factors affecting the segmentation point of the model in practical applications, such as resource utilization at the edge, system energy consumption, and network bandwidth. Therefore, it is crucial to take these factors into account in determining how to segment the neural network and solve the problem.

The main contributions of this paper are as follows.

1. Precisely determine the factors that affect model segmentation through pre-execution, and establish a cloud-edge collaborative computing paradigm based on model segmentation.
2. Construct a multi-objective model to obtain minimum completion time and maximum edge resource utilization for a training model in a cloud-edge collaborative environment.
3. Enhance the multi-objective solving algorithm by leveraging high concurrency and multiple populations while reducing the impact of uneven population distribution.
4. Use an enhanced multi-objective solving algorithm to calculate the optimal segmentation point of the model with the goal of minimizing cloud-edge collaboration completion time and maximizing resource utilization at the edge of model training, and then utilize the optimal segmentation point for model training in a cloud-edge collaboration environment.

Related works

The deployment of segmented neural network models in cloud-edge collaborative environments has been a widely discussed topic, both domestically and internationally. This issue directly affects task unloading time, resource allocation, system energy consumption, load balancing, and network bandwidth allocation in these environments. Kang et al. suggest that uploading a deep neural network model to the cloud for execution increases task completion time and mobile energy consumption. In this context, they proposed a segmentation method using the Neurosurgeon algorithm to reduce time and mobile energy consumption during model execution [18]. Kum et al. indicate that optimization methods for deploying AI applications at the edge potentially reduce AI accuracy. For this reason, they proposed a new deployment method that divides AI models into more than two parts and places them on either the edge or cloud using a container structure that converts AI into microservices. Despite increased end-to-end service delay, accuracy remained unchanged [19]. To achieve fast response time of convolutional neural networks (CNN) in practical applications, Zhang et al. proposed to compress and segment CNNs to generate a new network layer, which involves two steps. One is that the model's convolutional layer is dealt with using low-rank decomposition, followed by the full connection layer using singular value decomposition method, and the other is that processed network is segmented into fine granularity for minimum execution

delay. The model is fine-tuned to recover lost accuracy [20]. Hu et al. designed a DNN partition algorithm capable to identify the best segmentation point to minimize the overall delay of processing each frame under light load and maximize throughput under heavy load [21]. Mehta et al. put forward a new CNN segmentation method that considers network bandwidth, data size, and load at the edge to split the neural network model with the goal of minimizing network bandwidth consumption and combined it with various bandwidth optimization algorithms in spite of reduced final accuracy [22]. Yang et al. proposed a joint partitioning and compressing CNN method that effectively reduces the transmission delay of the model while ensuring accuracy, thereby reducing the inference time of the model [23]. Gao et al. designed a cloud-edge neural network model segmentation method composed of three outlets to transmit data, which greatly reduces end-to-end transmission delay compared to pure cloud computing [24]. Xue et al. presented a novel DNN partition method that segments the neural network model to minimize delay, cost, and energy consumption and escaped the failure caused by networks and other factors in the transmission process of large-scale computing data [25].

Neural network model segmentation has been widely used not only in cloud-edge collaborative systems but also in other distributed environments. Zhou et al. proposed a convolutional neural network acceleration framework to overcome the difficulty of deploying CNN on resource-constrained devices. This framework partitions CNN based on each layer's calculation and device communication delay and assigns each partition to corresponding devices [26]. Dey et al. proposed a CNN depth partitioning method to perform CNN tasks using limited computing equipment. Employing the input and output depth of the convolution layer to partition the load, the proposed method accelerates the computation of CNN tasks [27]. Qarariah et al. raised a DNN partition calculation graph strategy to train DNN models that cannot be trained in a single device, which involves clustering each layer of DNN into multiple partitions and then determining whether each partition breaks memory constraints with the aim to minimize end-to-end communication time and adjust the final partition according to memory constraints [28]. Kung et al. proposed to solve computationally intensive DNN reasoning on resource-constrained devices by utilizing the distributed deep neural network (DDNN) consisting of a small neural network model on the terminal device to extract data features and a large neural network in the cloud to receive data features and perform inference operations [29]. Mao et al. presented a locally distributed mobile computing system to partition neural network models without resource constraints and compress them without any loss while greatly accelerating DNN [30]. Ao et al. proposed an end-to-end distributed training framework that takes into account computing resources, model segmentation, and task placement to reduce the impact of dynamic resources on neural network model training and improve the training efficiency of the model [31]. Hou et al. put forward a strategy to divide neural network models into edge devices, taking into account network and equipment conditions, as well as CNN characteristics. Experiments show that the presented method improves the inference speed of CNN [32]. Jeong et al. designed a method to partition the neural network model based on the current communication situation in a distributed environment [33]. Aiming at the shortage of computing resources for edge devices, a load-balancing algorithm was raised by Miao et al. to segment the neural network model and assign it to different edge devices for task completion [34]. Liu et al. brought about a new neural network model partitioning method that fully considers network bandwidth and load at the edge [35]. He et al. designed a serial queue model to calculate the end-to-end delay, minimizing the segmentation of neural network models and thus making better use of mobile edge computing [36].

The traditional cloud computing only considers task offloading between mobile devices and cloud servers and often fails to fully utilize edge nodes [37–39]. While some methods achieve the minimum delay of real-time inferring after neural network model segmentation, the computing resources at the edge cannot be fully utilized. On the other hand, some methods maximize the utilization rate of computing resources at the edge but require more time for task completion, which is not conducive to solving delay-sensitive problems. Other methods fail to achieve the goal of improving the utilization rate of the edge and minimizing the task completion time in spite of their consideration about the consumption of network resources and the load balancing of servers. Therefore, this paper proposes a cloud-edge collaborative task scheduling method based on model segmentation by considering considers the utilization rate of resources at the edge, network condition, size of model output data, and calculation delay in the cloud-edge collaborative environment, which is able to obtain the best segmentation point for the cloud-edge collaborative computing neural network model while minimizing task completion time and maximizing edge-resource utilization in the cloud-edge collaborative environment.

System modeling and problem formalization

Using cloud computing traditionally for tasks of neural network model offloading often results in increased completion time due to the need to transfer large task

data, but accompanied by insufficient utilization of edge resources. To shorten the time for large neutral network models to complete tasks and improve the utilization of edge resources, this paper proposes a cloud-edge collaborative task scheduling method based on model segmentation with the objective of minimizing the completion time of task offloading and maximizing the utilization of computing resources at the edge in a neural network model training. To achieve this objective, factors including delayed task calculation, the transmission delay of intermediate data, and the resource utilization at the edge are taken into account. By analyzing these factors, the best segmentation point for the cloud-edge collaborative computing neural network model can be obtained to optimize the overall task scheduling process. Assuming that the amount of data uploaded by users to the edge is *num*, the neural network model to be trained has N layers (Table 1).

Task completion time

There are many factors affecting task completion time, including calculation delay and the transmission delay of

Table 1 Notations

Notations	Description
num	the amount of data uploaded by users to the edge
N	the number of layers in a neural network model
T_δ	task calculation delay on the cloud or edge
$\delta \in \{e, c\}$	edge node and cloud node respectively
T_{total}	transmission delay of intermediate data
T_c, T_e	cloud computing delay, edge computing delay
F	FLOPs of the neural network model
F_c, F_e	cloud computing and edge computing capability
F_δ	FLOPS of the edge or cloud server
$F = \{f_1, f_2, ..., f_n\}$	FLOPs of each layer of the neural network model
C_{in}	the number of input characteristic matrices
K_w, K_h	width and height of convolution kernel
C_{out}	the number of output characteristic matrices
w, h	width and height of output characteristic matrices
$FLOPs_{\text{cov}}, FLOPs_{fc}$	FLOPs of convolution and full connection layer
N_{In}, N_{Out},	input features, output features
N_{core}, H_c	cores number and frequency of the processor
N_{float}	floating-point operations per cycle of the processor
$T_e = \{t_{e,1}, t_{e,2}, ..., t_{e,n}\}$	computing delay of each layer at the edge node
$T_c = \{t_{c,1}, t_{c,2}, ..., t_{c,n}\}$	computing delay of each layer at the cloud node
$V_{trans}, V_{up}, V_{down}$	transmission rate, uplink rate, and downlink rate
O	the size of data to be transmitted
$O = \{o_0, o_1, ..., o_{n-1}\}$	the data output of each layer
$T_{up} = \{t_{up,0}, t_{up,1}, ..., t_{up,n-1}\}$	uplink delay of the output data of each layer
$T_{down} = \{t_{down,0}, t_{down,1}, ..., t_{down,n-1}\}$	downlink delay of the output data of each layer
$T_{trans} = \{t_{trans,0}, t_{trans,1}, ..., t_{trans,n-1}\}$	transmission delay of the output data of each layer
D_{size}	the space occupied by the corresponding data type
$(\dim_1, \dim_2, ..., \dim_m)$	the Tensor size output
$T_{total} = \{t_{total,0}, t_{total,1}, ..., t_{total,n-1}\}$	the transmission delay of the data of each layer
$T = \{t_0, t_1, ..., t_N\}$	task completion time
M_{total}	the total memory of the system
M_{wait}	system memory consumption without task execution
$M = \{m_0, m_1, ..., m_n\}$	memory consumption of each layer when running task
$M_{cost} = \{m_{\cos t,0}, m_{\cos t,1}, ..., m_{\cos t,n}\}$	memory consumption of each layer
$R_{memory} = \{r_{m,0}, r_{m,1}, ..., r_{m,n}\}$	memory occupancy of each layer
$split$	the location of the segmentation point

intermediate data. The task completion time function can be expressed as $T = f(T_\delta, T_{total})$, where $\delta \in \{e, c\}$, T_δ represents the task calculation delay on the cloud or edge, and T_{total} represents the transmission delay of intermediate data from the edge to the cloud.

The neural network model training is typically computationally intensive, which requires a large number of computational resources to process vast data. Thus, computing delay is a crucial factor that significantly affects the entire task unloading process. It is essential to consider the calculation delay of each layer in determining the optimal segmentation point of the neural network model. Therefore, it is necessary to develop an accurate method for calculating the computing delay of each layer of the neural network for optimizing a cloud-edge collaborative task scheduling method based on model segmentation.

To achieve a more accurate calculation delay of the neural network model, two indicators known as floating-point operations (FLOPS) and floating-point per second (FLOPS) are used in conjunction. By utilizing both FLOPS and FLOPS in tandem, it becomes possible to obtain a more comprehensive understanding of the computation requirements for a given neural network model, which, in turn, allows for more precise calculation of the calculation delay and ultimately enables the determination of the optimal segmentation point for the model. Pre-executing is used to estimate the amount of computation required for each layer and calculate the computing capacity of the server according to the processor index of the server in the cloud or at the edge.

Let the calculation delay function be $T_\delta = f(F, F_\delta)$, where $\delta \in \{e, c\}$, T_c, and T_e represent cloud computing delay and edge computing delay, respectively,F represents the FLOPs of the neural network model, and F_c and F_e represent cloud computing capability and edge computing capability, respectively. The specific calculation method is as follows.

Since the neural network model mainly focuses on the convolution layer and the full connection layer in the calculation process, it is of great importance to understand how to more accurately acquire the computing workload of the two layers.

Let the FLOPs of each layer of the neural network model be $F = \{f_1, f_2, ..., f_n\}$, which can be calculated by:

$$FLOPs_{\text{cov}} = [(C_{in} \times K_w \times K_h) + (C_{in} \times K_w \times K_h - 1) + 1] \times C_{out} \times w \times h \tag{1}$$

$$FLOPs_{fc} = [N_{In} + (N_{In} - 1) + 1] \times N_{Out} \tag{2}$$

where $FLOPs_{\text{cov}}$ is the FLOPs of convolution layer, C_{in} represents the number of input characteristic matrices, K_w and K_h respectively represent the width and height of convolution kernel, C_{out} is the number of output characteristic matrices, w and h respectively represent the width and height of output characteristic matrices.

$FLOPs_{fc}$ is the FLOPs of the full connection layer, N_{In} and N_{Out} respectively represent the number of input features and the number of output features.

Assuming the FLOPS at the edge node as F_e and the FLOPS at the cloud node as F_c, the calculation formula can be written as:

$$F_\delta = N_{core} * H_c * N_{float}, \delta \in \{e, c\} \tag{3}$$

where F_δ is the FLOPS of the edge or cloud server, N_{core} is the number of cores of the processor, H_c is the dominant frequency of the processor, and N_{float} is the number of floating-point operations per cycle of the processor.

According to the FLOPs $F = \{f_1, f_2, ..., f_n\}$ of each layer, the FLOPS F_e of the edge and the FLOPS F_c of the cloud, the computing delay $T_e = \{t_{e,1}, t_{e,2}, ..., t_{e,n}\}$ of each layer of neural network at the edge and the computing delay $T_c = \{t_{c,1}, t_{c,2}, ..., t_{c,n}\}$ at the cloud are calculated, where $(1 <= n <= N)$. The calculation formula is expressed as:

$$t_{\delta,i} = \frac{f_i}{F_\delta} (\delta \in \{e, c\}, i \in [1, \mathrm{N}]) \tag{4}$$

In cloud-edge collaborative environments, the cloud node is often distant from the edge node. Thus, uploading the model and data to the cloud can lead to significant transmission delays in the case of large amount of data transmitted and poor network condition. These factors can ultimately increase the total task completion time of the neural network model. To mitigate this issue and further reduce task completion time via optimal segmentation point calculation, it is necessary to consider the impact of transmission delays on required data.

There are many factors affecting transmission delay, such as data size, network conditions, and allocated bandwidth. In order to gain these factors, data output of each layer of the neural network is obtained by pre-executing the neural network model. Furthermore, current network transmission rate is calculated by performing a point-to-point network status query and analyzing system network configurations. Considering the impact of transmission delay of the required data during computation can help further reduce the task completion time of neural network models in cloud-edge collaborative environments.

Let the function of transmission delay be $T_{total} = f(V_{trans}, V_{up}, V_{down}, O)$, where V_{trans}, V_{up} and V_{down} respectively represent network transmission

rate, uplink rate, and downlink rate between the edge end and the cloud end, and O represent the size of data to be transmitted. The specific calculation method is expressed as:

Let the data output of each layer of the neural network be $O = \{o_0, o_1, ..., o_{n-1}\}$. The current network transmission rate V_{trans} between the edge and the cloud can be obtained through the ping command. The corresponding uplink rate V_{up} and downlink rate V_{down} can be obtained by querying the system network configuration through the grep command. The uplink delay $T_{up} = \{t_{up,0}, t_{up,1}, ..., t_{up,n-1}\}$, downlink delay $T_{down} = \{t_{down,0}, t_{down,1}, ..., t_{down,n-1}\}$ and transmission delay $T_{trans} = \{t_{trans,0}, t_{trans,1}, ..., t_{trans,n-1}\}$ of the output data of each layer of the neural network can be calculated, where $(1 <= n <= N)$. The calculation formula is expressed as:

where D_{size} is the space occupied by the corresponding data type, and $(\dim_1, \dim_2, ..., \dim_m)$ is the Tensor size output by the neural network model.

$$o_i = \frac{\dim_1 * \dim_2 * ... * \dim_m * D_{size}}{8} \tag{5}$$

$$t_{\varepsilon,i} = \frac{o_i}{V_\varepsilon} (\varepsilon \in \{up, trans, down\}, i \in [0, N-1]) \tag{6}$$

According to the uplink delay $T_{up} = \{t_{up,0}, t_{up,1}, ..., t_{up,n-1}\}$, downlink delay $T_{down} = \{t_{down,0}, t_{down,1}, ..., t_{down,n-1}\}$ and transmission delay $T_{trans} = \{t_{trans,0}, t_{trans,1}, ..., t_{trans,n-1}\}$ of the output data of each layer of the neural network, the transmission delay $T_{total} = \{t_{total,0}, t_{total,1}, ..., t_{total,n-1}\}$ of the data of each layer of the neural network from the edge to the cloud, or that of intermediate data corresponding to each segmentation point from the edge to the cloud can be calculawted, where $(1 <= n <= N)$. The calculation formula is expressed as:

$$t_{total,i} = f(T_{up}, T_{trans}, T_{down}) = t_{up,i} + t_{trans,i} + t_{down,i} \ (i \in [0, N-1]) \tag{7}$$

Using transmission delay and calculation delay, the completion time of the task at different segmentation points can be accurately estimated. Assuming the partition point to be $split(0 <= split <= N)$, the task completion time $T = \{t_0, t_1, ..., t_N\}$ corresponding to each partition point can be calculated according to the calculation delay $T_e = \{t_{e,1}, t_{e,2}, ..., t_{e,n}\}$ of each layer of a neural network at the edge, the calculation delay $T_c = \{t_{c,1}, t_{c,2}, ..., t_{c,n}\}$ of the cloud, and the transmission delay $T_{total} = \{t_{total,0}, t_{total,1}, ..., t_{total,n-1}\}$ of the intermediate data corresponding to each partition point from the edge to the cloud. The calculation formula is specifically expressed as:

$$t_k = f(T_{total}, T_e, T_c) = t_{total,split} + \sum_{i=1}^{split} t_{e,i} + \sum_{j=split+1}^{n} t_{c,j} (1 <= k <= N-1) \tag{8}$$

$$t_k = f(T_{total}, T_c) = t_{total,0} + \sum_{j=1}^{n} t_{c,j} (k = 0) \tag{9}$$

$$t_k = f(T_e) = \sum_{i=1}^{n} t_{e,i} (k = N) \tag{10}$$

Edge resource utilization

The goal of minimizing completion time is usually considered, while the utilization of edge resources is often overlooked in cloud-edge collaborative environments, which might be accompanied by edge resource wastes. Therefore, real-time edge-end resource conditions need to be considered for calculating the optimal segmentation point.

Assume that the total memory of the system is M_{total}, the memory consumption in the absence of task execution is M_{wait}, and the system memory consumption of each layer when running the neural network model task is $M = \{m_0, m_1, ..., m_n\}$, where $(0 <= n <= N)$. The memory consumption $M_{cost} = \{m_{\cos t,0}, m_{\cos t,1}, ..., m_{\cos t,n}\}$ of each layer of the neural network model can be calculated by:

$$m_{\cos t,n} = f(M, M_{\text{wait}}) = m_n - M_{\text{wait}} (0 <= n <= N) \tag{11}$$

The memory occupancy of each layer of the neural network model is$_{R_{memory} = \{r_{m,0}, r_{m,1}, ..., r_{m,n}\}(0 <= n <= N)}$, which can be calculated by:

$$r_{m,n} = f(M_{\cos t}, M_{total}) = \frac{m_{\cos t,n}}{M_{total}} * 100\% \ (0 <= n <= N) \tag{12}$$

Problem modeling

The resource utilization rate at the edge, network transmission delay, and task calculation delay will affect the calculation of the segmentation point. Therefore, it is a multi-objective optimization problem to calculate segmentation points, which includes minimizing task completion time and maximizing memory resource

utilization at the edge. The multi-objective function can be constructed as:

$$\begin{cases} \min\{T\} \\ \min\{1 - R_{memory}\} \end{cases} \quad \begin{cases} C1 : 0 <= split <= N \\ C2 : split \in \mathbb{Z} \\ C3 : M_{cost} + M_{\text{wait}} <= M_{total} \\ C4 : 0 <= R_{memory} <= 1 \end{cases} \tag{13}$$

where *split* is the location of the segmentation point, R_{memory} is the memory resource utilization at the edge, and T is task completion time.

The NSGA-III (non-dominated sorting genetic algorithm-III) algorithm is modified to achieve the optimal segmentation point. The NSGA-III algorithm has the advantages of great running speed, good convergence, and nice high-dimensional multi-objective optimization results. There are four steps in the process of retaining individuals by the method, which are reference point generation, population adaptive standardization, association between individuals and reference points, and individual selection. When the population iterates a certain number of times, the optimal multi-objective optimization solution set will be generated. However, the NSGA-III algorithm also has certain limitations. Due to the high randomness in the selection of initial reference points, the initial state of the population will not be excellent enough, thereby affecting the final results. To solve this problem, multi-population multi-objective optimization methods have emerged. The multiple populations can be combined with the NSGA-III algorithm to avoid the impact on the final results caused by the poor initial state of the population. At the same time, a combination of multi-threading and multi-group multi-objective (MTMGMO) optimization is used to reduce calculation time.

The number of populations is set as p, the initial number of populations for each population as $Pop = \{pop_1, pop_2, ..., pop_p\}$, and the thread group as $Thd = \{thd_1, thd_2, ..., thd_p\}$. In the MTMGMO algorithm, multiple threads are used to avoid increasing the required time caused by the serial computing for multiple groups. Starting a new thread for each population not only reduces the impact on the final result caused by the poor initial state of the population but also avoids the long serial computing time for multiple populations.

Cecms algorithm

Taking the training of the neural network model required by end users as an example, a cloud-edge collaborative task scheduling method based on model segmentation is shown in Fig. 1.

A cloud-edge collaborative task scheduling method based on model segmentation involves three steps, namely, pre-executing, optimal segmentation point

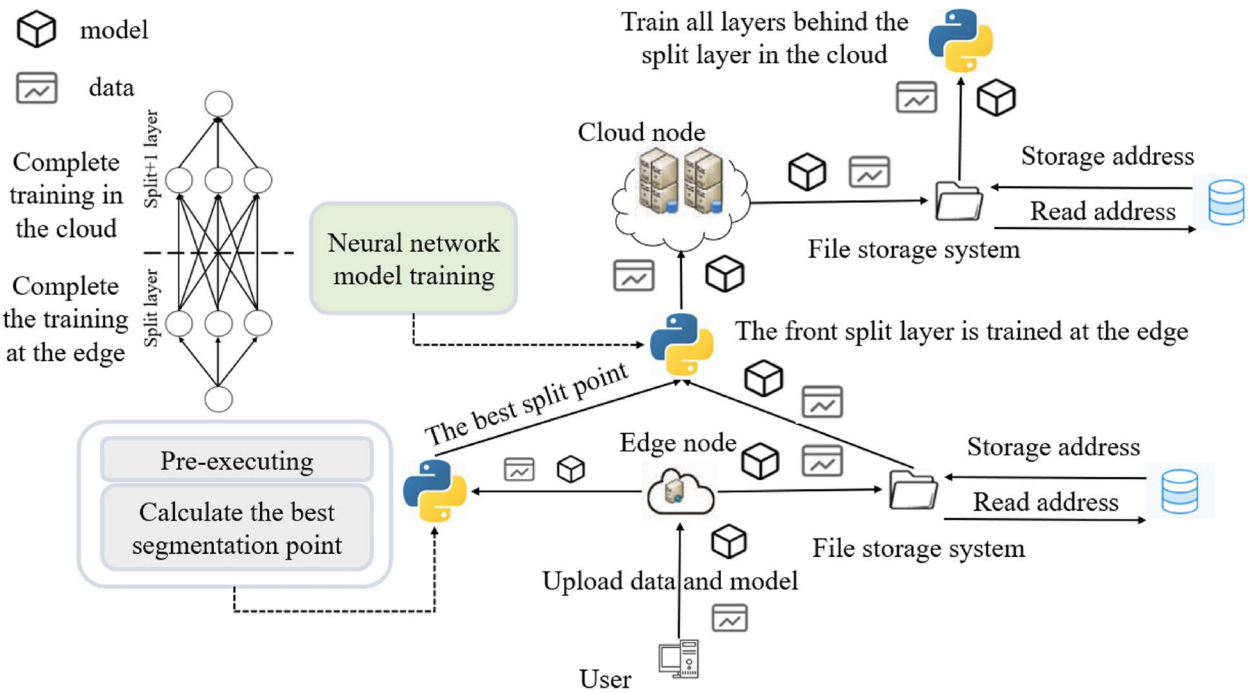

Fig. 1 Flow chart of cloud-edge collaborative task scheduling method based on model segmentation

calculation, and cloud-edge collaborative training. Their relationships are illustrated in Fig. 2.

Pre-executing is performed at the edge node to calculate the FLOPs of each layer of the neural network model, the amount of data output, and the computing resource occupancy at the edge node. The pre-executing process is demonstrated in Fig. 3.

The specific steps are as follows:

Firstly, the user uploads his or her training data and neural network model to the edge node. The edge node then stores the necessary data and model files in the file storage system and records the storage address in the database. Secondly, the edge node loads the neural network model and data required for training by referencing the saved address information in the database. The system then evaluates whether all of the data can be used for pre-executing based on the resource threshold of edge node. If the computing resources available at the edge exceed the pre-executing threshold and there is no

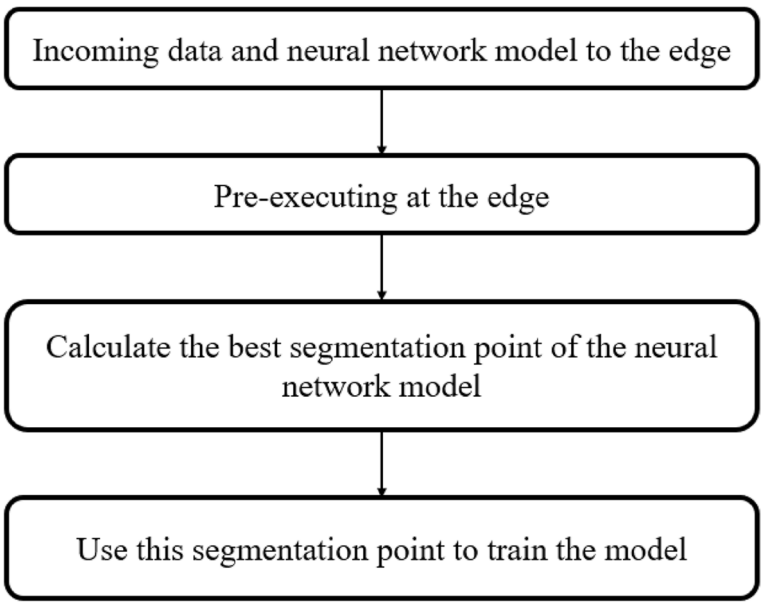

Fig. 2 Schematic diagram of the relationship between Pre-executing, optimal segmentation point calculation, and cloud edge collaborative training

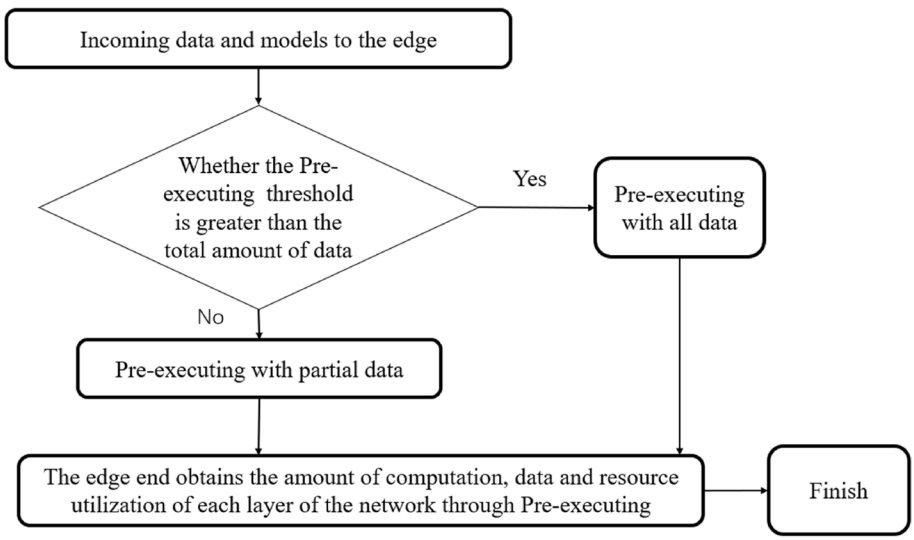

Fig. 3 Pre-executing flow chart

limit, then the system will proceed to pre-train the entire dataset.

However, if the data exceeds the threshold, the system uses a sample of 10% of the data to calculate the FLOPs of each layer of the neural network model and evaluate the computational resource occupancy of the edge node, thus allowing the acquisition of the information required for optimal resource allocation and preventing the overconsumption of computing resources. By selectively training only a portion of the data, the edge node can balance the computation workload while still accurately calculating the FLOPs of each layer of the neural network model, the amount of data output, and the computing resource occupancy at the edge node. Overall, this approach ensures that the edge node can effectively take advantage of its computing resources while providing reasonable parameters for calculating segmentation points.

The edge node performs the best segmentation point calculation algorithm to calculate the best segmentation point of the neural network model. The calculation process is shown in Fig. 4.

The specific steps are as follows.

1: The edge node calculates the computing delay of each layer of a neural network at the edge node and the computing delay at the cloud node according to the FLOPS of each layer, edge node, and cloud node.
2: The edge node obtains the current network transmission rate between the edge node and the cloud through the ping command, and queries the system network configuration through the grep command to obtain the corresponding uplink rate and downlink rate.
3: The edge node calculates the uplink delay, downlink delay, and transmission delay of the output data on each layer of the neural network according to the data output, current network transmission rate, uplink rate, and downlink rate.
4: The edge node calculates the transmission delay of each layer of neural network data from the edge node to the cloud node according to the uplink delay, downlink delay, and transmission delay of output data of each layer of the neural network, or that of intermediate data corresponding to each segmentation point from the edge node to the cloud node.
5: The edge node obtains the total time delay trained by the neural network model corresponding to each segmentation point according to the calculated time delay of each layer of a neural network at the edge node and the calculated time delay of the cloud node and the transmission time delay of the intermediate data corresponding to each segmentation point from the edge node to the cloud node.
6: According to the total time delay corresponding to each segmentation point and the computing resource occupancy rate of the edge node during the execution of each layer of the neural network; the MTMGMO algorithm is used for multi-objective optimization to obtain the best segmentation point of the neural network model.

After calculating the best segmentation point of the model, the model is segmented according to the best segmentation point. The network layer before and

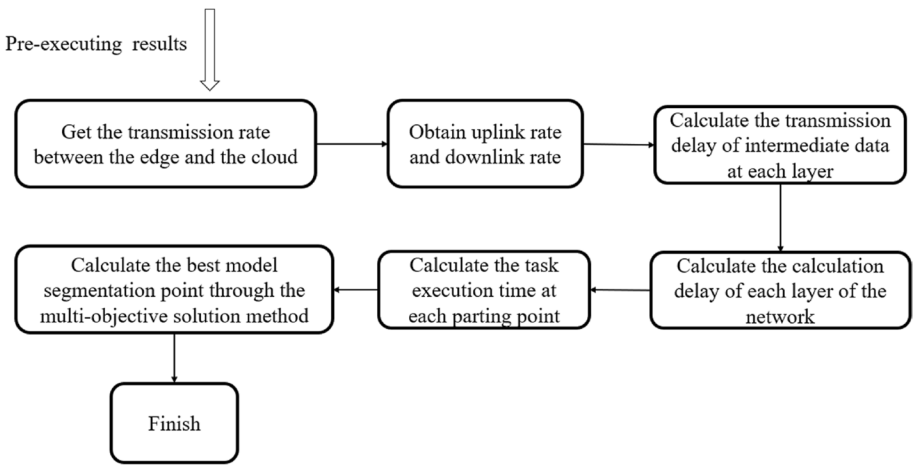

Fig. 4 Flow chart of calculating the best segmentation point

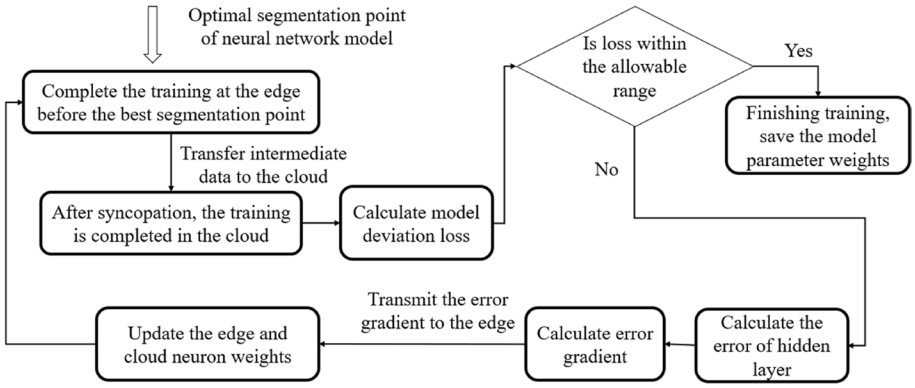

Fig. 5 Training process of the neural network model in cloud edge collaborative environment

after the segmentation point is respectively trained at the edge and cloud. The training process is shown in Fig. 5.

The specific steps are as follows.

1: The edge node uploads the data needed for training and the neural network model needed for training to the cloud.
2: The cloud stores the neural network model to be trained and the data to be trained in the file storage system, and saves the storage address in the database.
3: The edge and cloud nodes load neural network models and all training data.
4: The edge node trains the network layer before the segmentation point to obtain intermediate data.
5: The edge node transmits the intermediate data to the cloud through the Socket communication system.
6: After receiving the intermediate data, the cloud node executes the neural network layer after the segmentation point to obtain the final result. Calculate the deviation loss between the final and expected results, and determine whether the deviation loss is within an acceptable range. If it is within the acceptable range, save the model parameter weight to end the training, and if it is not within the acceptable range, calculate the neuron error of the hidden layer.
7: The cloud calculates the error gradient according to the neuron error of the hidden layer. The error gradient is transmitted to the edge through the Socket communication system, and then the cloud updates the cloud neuron weight according to the error gradient. In this process, the cloud error gradient is firstly transmitted to the edge, and then the cloud and the edge update the parameters of the neural network model.
8: The edge receives the error gradient transmitted from the cloud and updates the edge neuron weight according to the error gradient. In this process, the error is firstly transmitted from the back to the front, and then the weight and offset value from the back to the front is modified according to the error.
9: Proceed to next training, which is to return to Step 4.

Necessary conditions for model training only in the cloud or edge:

(1) If the computing resources at the edge fail to the requirements for training the first layer of the neural network, all the training will be conducted in the cloud, and the segmentation point is 0.
(2) If all models are trained at the edge, training in the cloud will lead to insufficient utilization of computing resources at the edge. At this time, the segmentation point is at the last layer of the network.
A cloud-edge collaborative task scheduling method based on model segmentation is designed as follows.

Algorithm 1 A cloud-edge collaborative task scheduling method based on model segmentation

Input: model, traindata, threshold, population;

Output: transmission delay, task completion time, utilization of memory resources of edge nodes, best model segmentation point, loss;

01: def function Pre-executing(traindata, model):

02: Obtain C_{in}, K_w, K_h, C_{out}, w, h, N_{In} and N_{Out} based on the model;

03: Calculate the FLOPs of each layer $F=\{f_1,f_2,...,f_n\}$ base on C_{in}, K_w, K_h, C_{out}, w, h, N_{In} and N_{Out}; // according to the above formula (1) and (2)

04: data = get_ data_size(traindata, model_info) // Calculate the output size for each layer $O=\{o_0,o_1,...,o_{n-1}\}$ and obtain M_{total}, M_{wait}, $M=\{m_0,m_1,...,m_n\}$ $M_{cost}=\{m_{\text{cost},0},m_{\text{cost},1},...,m_{\text{cost},n}\}$

05: Calculate the memory occupancy of each layer $R_{memory}=\{r_{m,0},r_{m,1},...,r_{m,n}\}$ base on M_{total}, M_{wait}, $M=\{m_0,m_1,...,m_n\}$ and $M_{cost}=\{m_{\text{cost},0},m_{\text{cost},1},...,m_{\text{cost},n}\}$, // according to the above formula (11) and (12)

06: return $F=\{f_1,f_2,...,f_n\}$, data, $R_{memory}=\{r_{m,0},r_{m,1},...,r_{m,n}\}$

07: end function

08: if (threshold > traindata.size()):

09: computation, data, utilization=Pre-executing(all traindata, model);

10: else:

11: computation, data, utilization=Pre-executing(partial traindata, model);

12: End if

13: transrate = ping(IP, port); // Get the transmission rate V_{trans} between the edge and the cloud

14: uprate, downrate=getUpAndDown(); // Obtain uplink rate V_{up} and downlink rate V_{down}

15: Obtain N_{core}, H_c and N_{float} base on edge or cloud node

16: caldelay = getCalculationDelay(computation); // computing calculation delay $T_e=\{t_{e,1},t_{e,2},...,t_{e,n}\}$ and $T_c=\{t_{c,1},t_{c,2},...,t_{c,n}\}$ base on formula (3) and (4)

17: trandelay = getTransmissionDelay(transrate, uprate, downrate, data); // computing transmission delay $T_{total}=\{t_{total,0},t_{total,1},...,t_{total,n-1}\}$ base on formula (5), (6) and (7)

18: exeTime = (trandelay, caldelay); // computing task execution time $T=\{t_0,t_1,...,t_N\}$ base on formula (8), (9) and (10)

19: for i in range(population):

20: result = thread(MTMGMO(exeTime, utilization)) // Multi-objective solving for formula (13)

21: split = GetBestSplit(); // Calculate the best model segmentation point

22: While (loss < Loss target): // Traing

23: mid = Edge(model, data, split); // training on edge

24: Transfer(mid, cloud); // to cloud

25: result = cloud(model, mid, split); // training on cloud

26: loss(result, data);

27: gradient = loss.gradient;

28: Transmit(gradient, edge); // to edge

29: EdgeUpdate(gradient); // update parameters on edge

30: cloudUpdate(gradient) // update parameters on cloud

31: End while

Finally, this paper compares the time complexity of CECMS algorithm with an adaptive DNN inference acceleration framework with end–edge–cloud collaborative computing algorithm [40] (ADC) and a Dynamic Adaptive DNN Surgery method (DADS) [21]. The comparison results are shown in Table 2.

The time complexity of the CECMS algorithm proposed in this paper mainly originates from the multi-objective solving part and the model pre-execution with a small portion of data. The multi-objective solving algorithm is an improvement based on NSGA-III, so the time complexity of the multi-objective solving part is $O\ (n^2 log\ n)$, where n is the number of population. When the model pre-executes a small portion of data, the time complexity is mainly related to the size of the convolution kernels and input/output channels in each convolutional layer. Assuming a neural network has N layers, where L layers are convolutional layers, each convolutional layer has a kernel size of k_i, an output matrix size of m, and the input and output channels are in_i and out_i, respectively, then the time complexity of the model's pre-execution with a small portion of data is $O(\sum_{i=1}^{L} m^2 * k_i^2 * in_i * out_i)$. In summary, the time complexity of the CECMS algorithm *is* $O\ (\sum_{i=1}^{L} m^2 * k_i^2 * in_i * out_i) + O\ (n^2 log\ n)$, higher than that of DADS algorithm, because of its use of multi-objective solving algorithm to calculate the optimal segmentation point.

Experiments and analysis

A series of experiments were carried out to verify the effectiveness of a cloud-edge collaborative task scheduling method based on model segmentation. It was assumed that there was an edge node and a cloud node in the experimental environment. The cloud node was a GPU cloud server with a specification of 16-core 60G and Nvidia A100 GPU. The edge node was a cloud server with 8 cores and 16G that used Cifar-10 data-set, and the fine-tuned AlexNet [41] model and VGG [42] model as the neural network models. There were 13 hidden layers in the AlexNet model and 34 hidden layers in the VGG model. During pre-execution, 5,000 images randomized from the Cifar-10 dataset were used as input to the AlexNet model and the VGG model. The Cifar-10 dataset was divided into 10 batches in the training process, each containing 5,000 images. In the multi-objective solving process, four populations were initialized, with each initial population size of 100, 200, 300 and 400. The number of iterations was 500.

Table 2 Time complexity comparison

	CECMS	ADC	DADS
Time complexity	$O(\sum_{i=1}^{L} m^2 * k_i^2 * in_i * out_i) + O\ (n^2 log\ n)$	$O(N^4)$	$O(N^3)$

To verify the effectiveness of the MTMGMO algorithm, comparative experiments were conducted with a single-threaded multi-population algorithm. Four populations were initialized with each initial population size of 100, 200, 300, and 400. The number of iterations was set to 500. Tables 3 and 4 demonstrate the MTMGMO algorithm and the single-threaded multi-population algorithm, respectively, in which the "Initial distribution" column represents the initial population distribution, and the "Final distribution " column represents the population distribution after 500 iterations. It can be observed from both tables that the final results of two algorithms were identical in the case of the same initial population.

Regarding execution time, experiments were conducted for ten times on both algorithms in the same environment, as shown in Table 5 and 6. According to the experimental results, the proposed MTMGMO algorithm was 60% faster than the single-threaded multi-population algorithm. Overall, these comparative experiments demonstrated that the MTMGMO algorithm was superior to the single-threaded multi-population algorithm in terms of both execution speed and optimization accuracy, which highlighted the effectiveness of the proposed MTMGMO algorithm.

To ensure the correctness of the optimal segmentation points of the AlexNet and VGG models, the experiments were conducted on each possible segmentation point of the model, and the task execution time and edge node memory resource utilization were obtained under different segmentation points. The experimental results are shown in Fig. 6. The abscissa represents each possible segmentation point of the model. Since the VGG model after the 15th layer potentially caused the required memory of the edge nodes to exceed the system memory capacity, it was not possible to segment points. Figure 6a and b respectively show the task execution time and the memory resource utilization of edge node corresponding to each possible segmentation point in the AlexNet and VGG models.

Figure 6c more intuitively illustrates the impact of data transmission delay and task execution time on the segmentation point of the AlexNet model. As can be seen from the figure, data transmission delay decreased while task execution time increased layer by layer in the first six layers, which was attributed to the decline in layer-by-layer calculation of the neural network, the amount of data output by the neural network model but the increase in the calculation amount. Compared with the 6th layer, despite reduced transmission delay, task calculation delay increased. Therefore, task execution time was the shortest when slicing in the 6th layer. It can be seen from

Table 3 Population distribution of MTMGMO algorithm

Discrete values of the population domain	0	1	2	3	4	5	6	7	8	9	10	11
Initial distribution	82	104	78	78	83	66	75	75	87	97	86	89
Final distribution	0	0	0	0	0	165	165	167	167	168	0	168

Table 4 Population distribution of single threaded multi population algorithms

Discrete values of the population domain	0	1	2	3	4	5	6	7	8	9	10	11
Initial distribution	82	104	78	78	83	66	75	75	87	97	86	89
Final distribution	0	0	0	0	0	165	165	167	167	168	0	168

Table 5 MTMGMO algorithm execution time

Number of executions	1	2	3	4	5	6	7	8	9	10
Execution time	1.743	1.684	1.452	1.452	1.801	2.026	1.633	1.674	1.930	1.855

Table 6 Single threaded multi swarm algorithm execution time

Number of executions	1	2	3	4	5	6	7	8	9	10
Execution time	3.625	3.589	3.598	3.599	3.581	3.585	3.582	3.599	3.612	3.595

Fig. 6a that the resource utilization of edge nodes gradually increased with the increase in the number of layers increases (except for the 2nd layer) when the memory resource utilization of edge nodes was taken into account and that the optimal segmentation point of the model was on the 6th layer when comprehensively considering the utilization of edge node memory resources and task execution time. Prior to the 6th layer, task execution time decreased while the resource utilization of edge nodes increased layer by layer. After the 6th layer, both of the two increased, which, however, was not conducive to task offloading.

Figure 6d clearly visualizes the impact of data transmission delay and task execution time on the segmentation points of the VGG model. As demonstrated in Fig. 6d, slicing on the 5th or 10th layer yields the shortest task execution time. Concerning the memory resource utilization of edge node, resource utilization rate gradually increased as the number of layers increased starting from the 2nd layer, as shown in Fig. 6b. When the memory resource utilization of edge node and task execution time were evaluated simultaneously, it was obvious that the optimal segmentation point of VGG model was on the 10th layer. Though task execution time was the shortest on the 5th layer, the resource utilization rate of edge nodes was insufficient.

These findings provide valuable guidance for selecting the optimal segmentation points of the VGG model and the AlexNet model. They serve to ensure effective task offloading between the cloud node and the edge node, while reducing overall task execution time and maximizing resource utilization rate.

Through pre-execution, the transmission delay of each layer of the neural network in the AlexNet model and the VGG model, task execution time at the edge, and the memory resource utilization at the edge nodes were obtained. To analyze and compare the changes in each layer more clearly, the data were normalized, as shown in Figs. 7 and 8, where the abscissa indicates the number of layers in the neural network model. It should be noted that task completion time was composed of the sum of both transfer time and task execution time.

An analysis of the two graphs in Fig. 7 indicates that to minimize task completion time alone, the optimal segmentation point for the AlexNet model should be set on the 6th layer. However, there were several options for selecting optimal segmentation points for the model, considering minimizing task completion time and

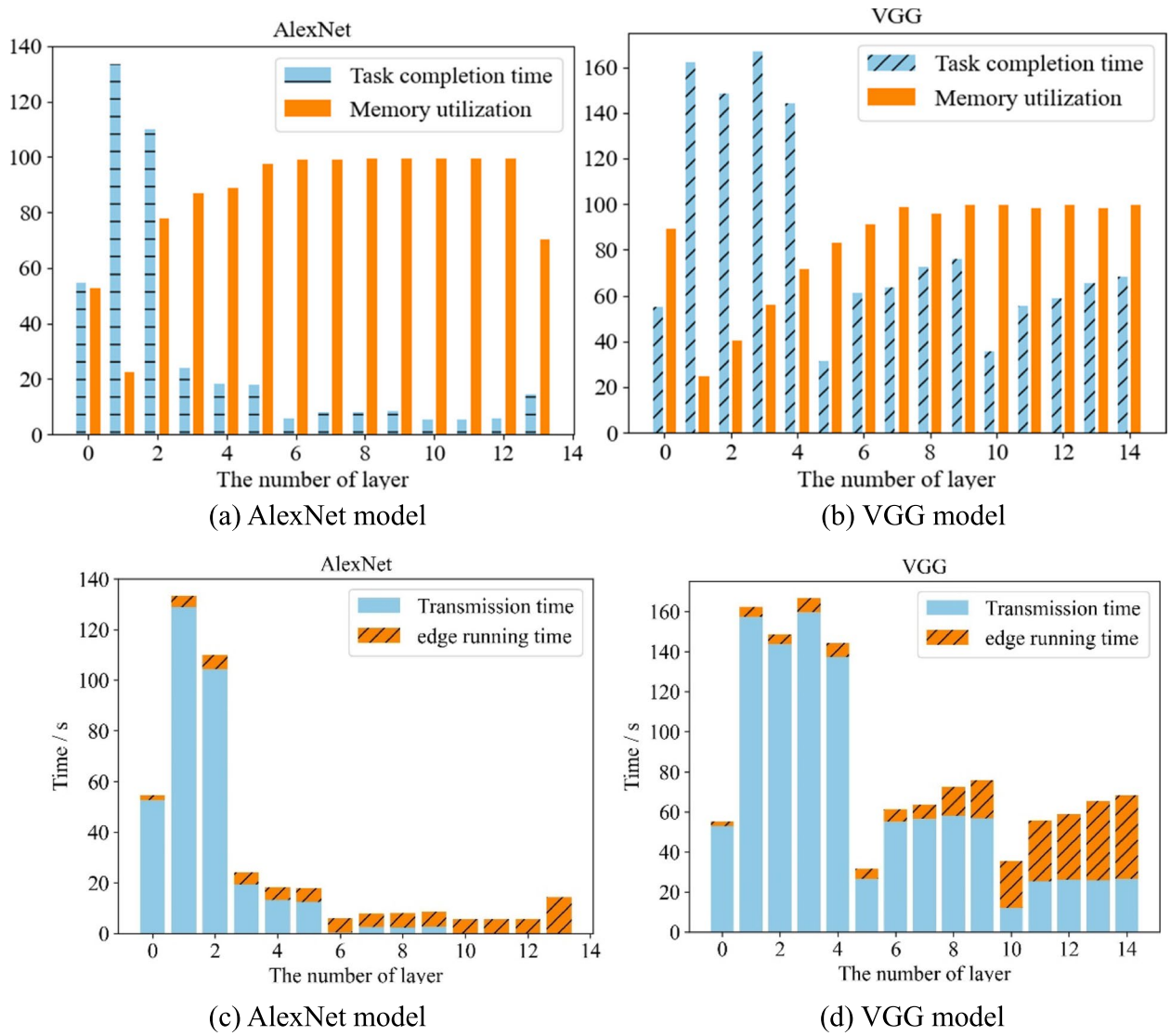

Fig. 6 The transmission delay, task completion time, and utilization of memory resources of edge nodes of each layer of neural network in AlexNet or VGG model

maximizing the utilization of edge memory resources (i.e. minimizing the remaining edge memory resources). In order to calculate the optimal segmentation point more accurately, the multi-objective solution algorithm MTMGMO was utilized for final selection.

The optimal segmentation point of the AlexNet model was determined to be on the 6th layer using the multi-objective solution algorithm MTMGMO. Subsequently, the model was trained with the edge executing the first six layers of the neural network model and the cloud executing all subsequent layers. The training effect of the model is presented in Fig. 9. At the same time, experiments were also conducted with the segmentation point on the 3rd layer and cloud-only training. By firstly determining the minimum task completion time and then adjusting the resource utilization of edge without using the multi-objective solution algorithm, the segmentation point was calculated to be on the 3rd layer. The results showed that such segmentation point was not the optimal, which further indicates the need to consider multi-objective optimization.

Figure 9 demonstrates that when the optimal segmentation point was on the 6th layer, the loss curve firstly decreased but eventually fell within a range similar to that when experiments were performed using the segmentation point on the 3rd layer or cloud-only training after numerous training sessions. However, despite increased training sessions, training duration was

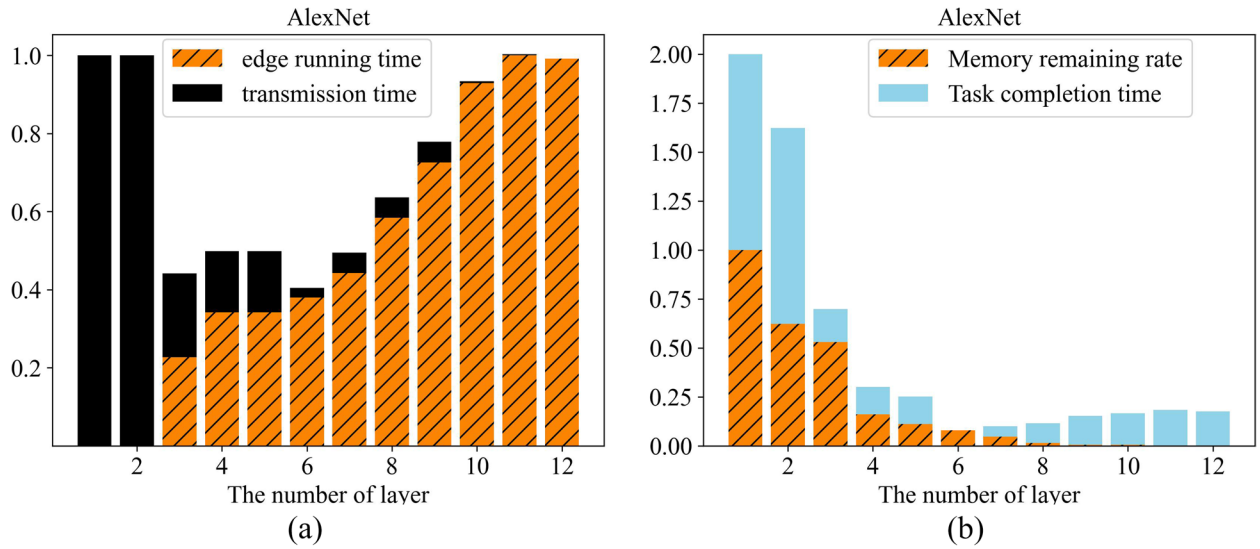

Fig. 7 The completion time of tasks corresponding to each split point of AlexNet and the Memory remaining rate at the edge

significantly reduced by 75% compared to that when experiments were performed using the segmentation point on the 3rd layer, and by 85% compared to that when experiments were performed using cloud-only training, which could primarily be attributed to the large amount of data uploaded to the cloud in a pure cloud environment, resulting in substantial transmission delay.

As shown in Fig. 8, considering both the minimization of task completion time and the maximization of memory resource utilization at the edge, the optimal segmentation point on the 10th layer was selected for the VGG model. In order to accurately determine the optimal segmentation point, MTMGMO was used to select the segmentation point consistent with the image analysis, and the optimal segmentation point of the VGG model was determined to be on the 10th layer.

Upon completion of the optimal segmentation point calculation, the model training task was commenced. The edge executed the first 10 layers of the neural network model, while the cloud was responsible for executing all subsequent layers. Furthermore, cloud-only model training was also performed for comparison. The training effect of the model is presented in Fig. 10.

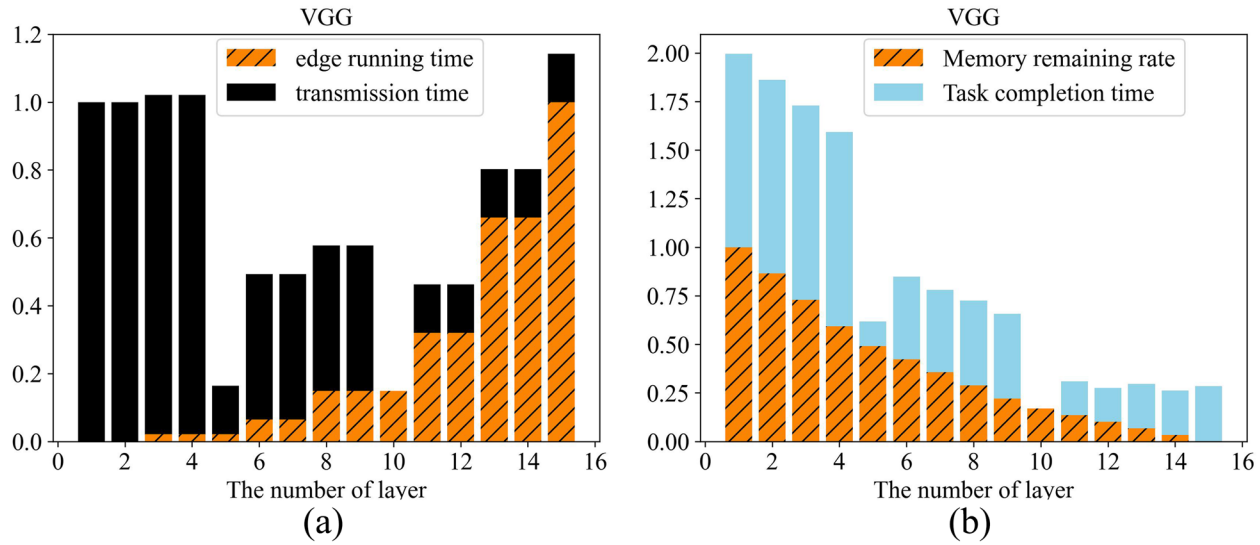

Fig. 8 The completion time of tasks corresponding to each split point of VGG and the Memory remaining rate at the edge

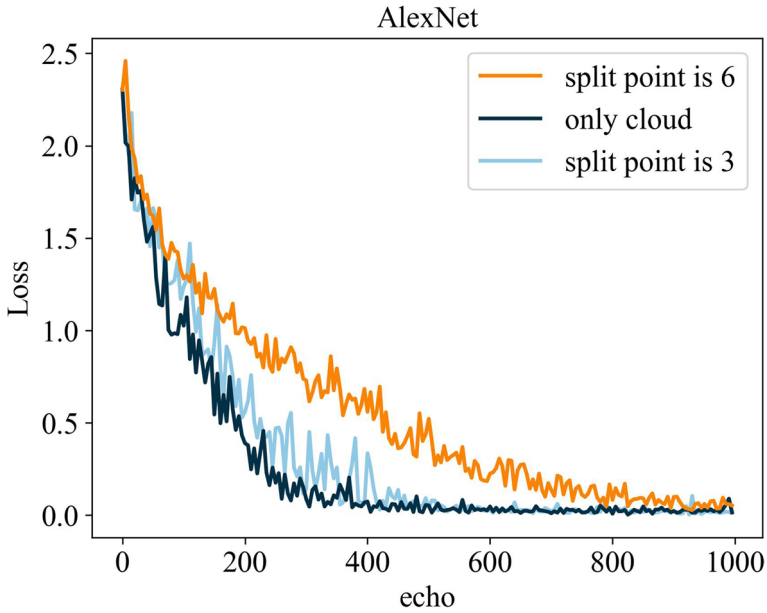

Fig. 9 The Loos curve of the AlexNet model changes

Based on the above analyses, the optimal segmentation points of the AlexNet model and the VGG model calculated using the method proposed in this paper were consistent with the actual results.

In addition, for the purpose of further demonstrating the effectiveness of the method proposed in this paper, comparative experiments were conducted using an adaptive DNN inference acceleration framework with end–edge–cloud collaborative computing algorithm (ADC) [40] and a Dynamic Adaptive DNN Surgery method (DADS) [21]. DADS was employed to segment the neural network model, intending to minimize overall delay based on different network conditions. The AlexNet and VGG16 models were selected for the comparative experiment by DADS method. The optimal segmentation point calculated using the DADS method

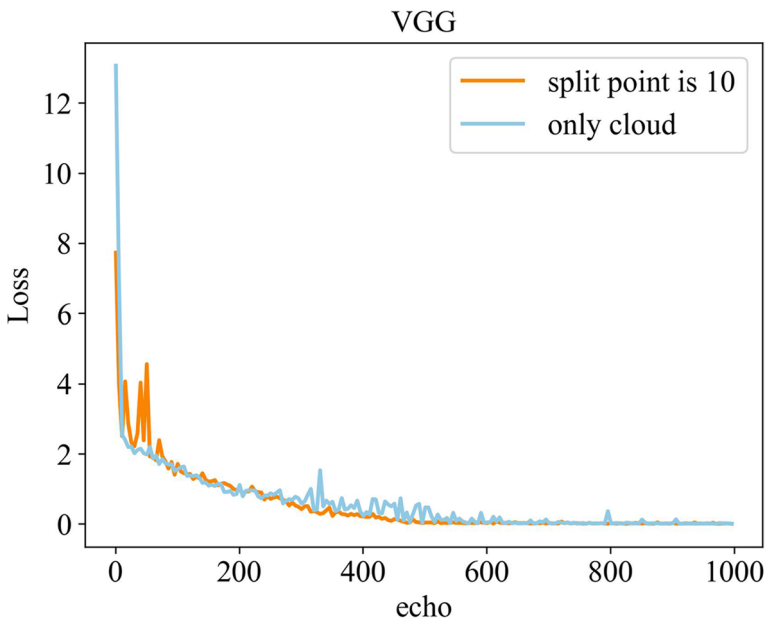

Fig. 10 The Loos curve of VGG model changes

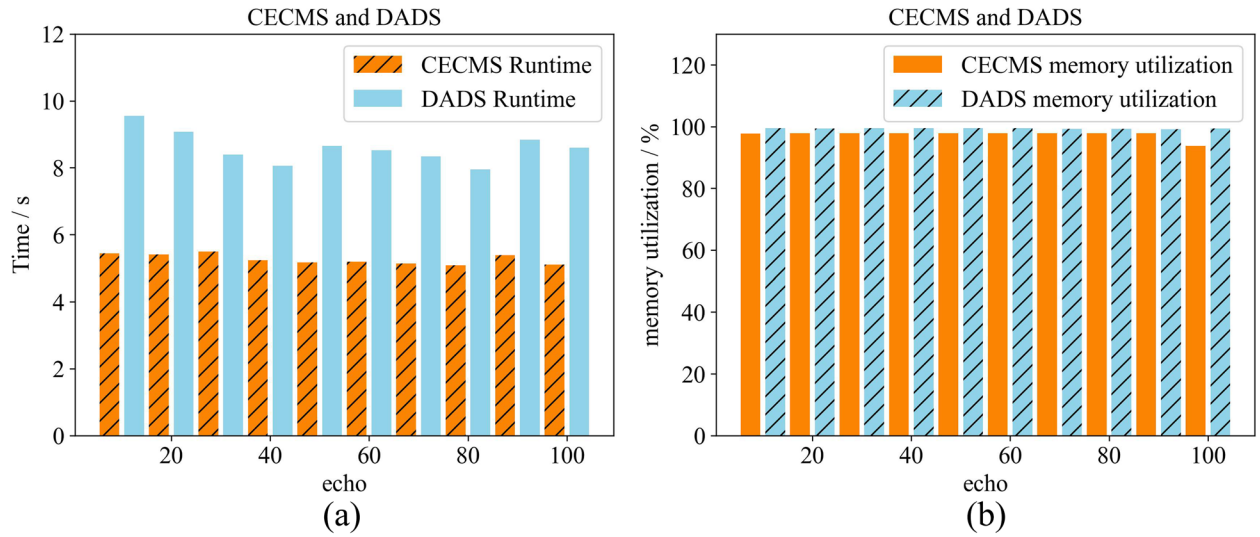

Fig. 11 Comparison of AlexNet between CECMS and DADS

respectively fell on the 9th layer of the AlexNet model and on the 18th layer for the VGG16 model, while that calculated using the CECMS method on the 6th layer for the AlexNet model and on the 14th layer for the VGG16 model. The comparative experimental results of AlexNet and VGG16 are shown in Figs. 11 and 12. In terms of the AlexNet model, the task completion time calculated by the CECMS method in the case of the segmentation point on the 6th layer was shorter than that calculated by the DADS method in the case of the segmentation point on the 9th layer, but with little difference in edge memory utilization between the two, as shown in Fig. 11. For the VGG16 model, the task completion time obtained by the CECMS method was basically the same as that by the DADS method, and the fluctuation in numerical values was caused by real-time network fluctuation. Meanwhile, the edge memory utilization acquired by the CECMS method was only about 1% lower than that by the DADS method, proving the effectiveness of the CECMS method.

To verify the effectiveness of the CECMS method on non-public datasets, a convolutional neural network, named MedicalNet, with 13 layers, as shown in Fig. 13, and a pathological recognition dataset of 20,000 images were used for experiments. This paper found that the recognition rate of medicalNet did not change significantly when the image was rotated 90 or 180 degrees

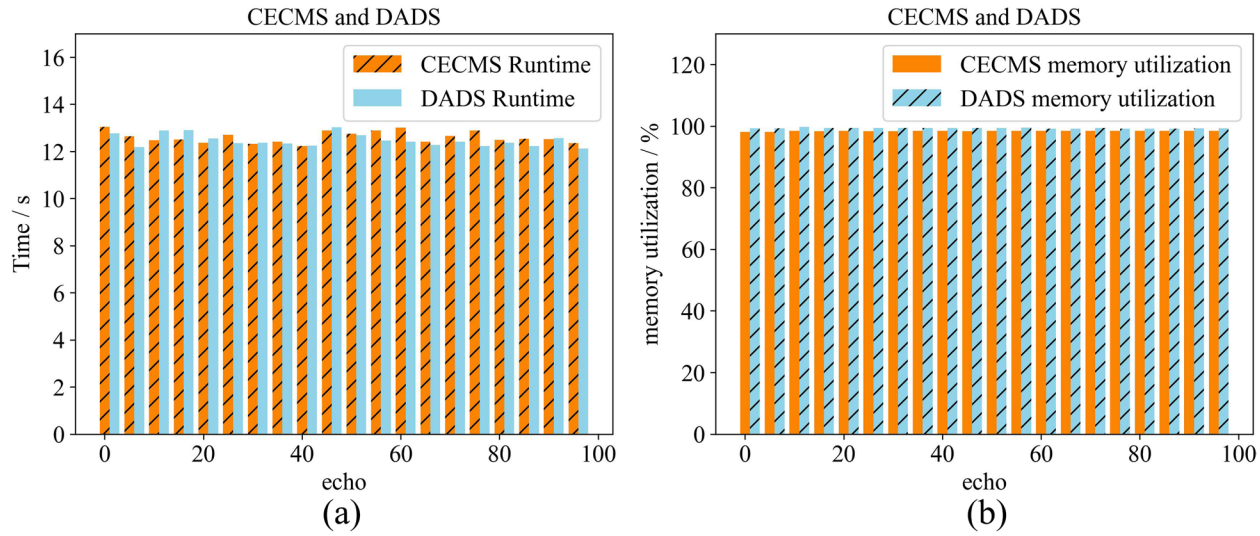

Fig. 12 Comparison of VGG16 between CECMS and DADS

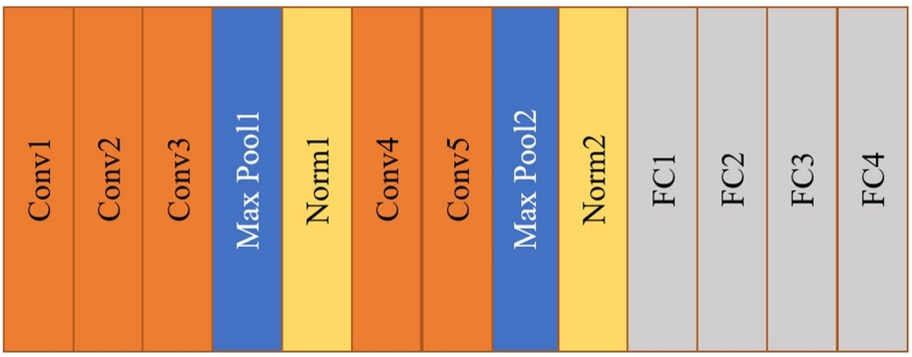

Fig. 13 The structure of MedicalNet

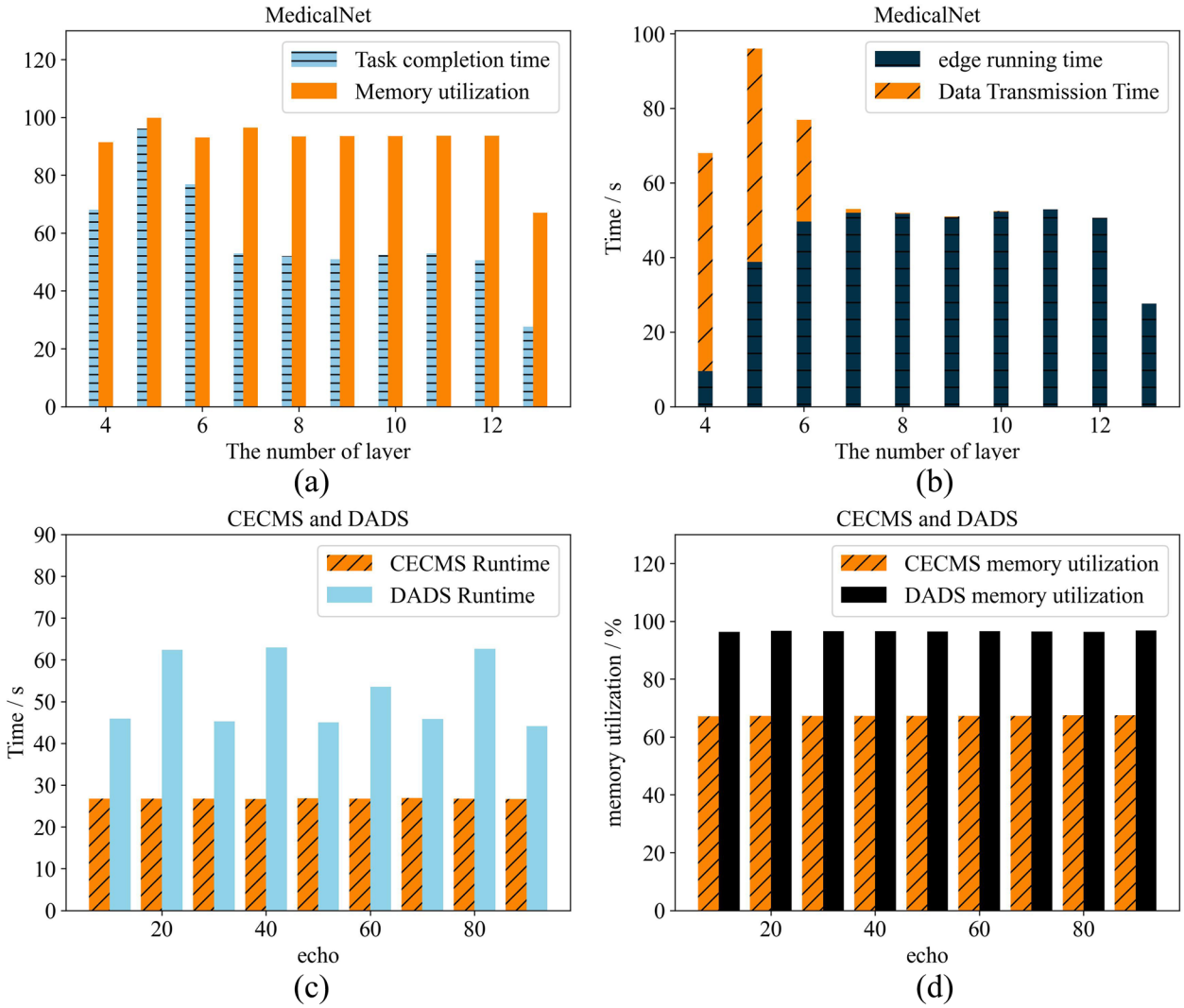

Fig. 14 The completion time and the Memory remaining rate of tasks corresponding to each split point of MedicalNet, as well as the comparison of MedicalNet between CECMS and DADS

through a series of experiments on model sensitivity. However, medicalNet's recognition rate of images decreases when images are cropped. It is because that the rotation operations do not change the relative position of key factors in the image, while the cropping operations may lead to the loss of key factors in the image. The optimal segmentation point calculated using the CECMS method was on the 13th layer, meaning that all the asks were unloaded at the edge, while that calculated using the DADS method was on the 8th layer. The task completion time and edge memory resource utilization of each layer in the model are shown in Fig. 14. As can be seen from the figure, memory resources at the edge were sufficient for task calculation, and task completion time was the least when the optimal segmentation point was set on the 13th layer. Thus, tasks should not be unloaded to the cloud for execution. The memory resource utilization of the edge node was the maximum when the optimal segmentation point was set on the 8th layer, and its calculation time was slightly longer than that when the optimal segmentation point fell on the 13th layer.

The ADC algorithm is a segmentation algorithm aimed at minimizing end-to-end inference latency. This algorithm finds the segmentation point that minimizes end-to-end inference latency by traversing all layers of the neural network. The neural network models AlexNet and VGG with the same dataset Cifar-10 were used for this comparison. This paper transforms the cloud-edge-end strategy to the cloud-edge strategy so that the ADC algorithm conforms to the cloud-edge collaboration scenario, which means that the first segmentation point in the ADC algorithm is set to 0.

A total of 100 batches (i.e. echo), were used in the comparative experiment between CECMS and ADC, with each set to 5,000 images. The comparative experimental results of the AlexNet model are shown in Fig. 15. The segmentation points calculated by the two are 10 and 6, respectively.

Through multiple experiments, it was found that it takes approximately the same amount of time for AlexNet to complete the task when the segmentation points are 6 and 10, but with certain differences in memory resource utilization at edge node. When the segmentation points are 6 and 10, task runtime is 5.25 s and 5.45 s, respectively, and edge memory resource utilization rate is 97.3% and 99.5%, respectively.

In the absence of the large difference in the task runtime when segmentation points 6 and 10, the CECMS algorithm tends to select a segmentation point of 10 for calculation due to its higher memory resource consumption, aiming to maximize the use of edge resources and reduce cloud load, while the ADC algorithm tends to choose a segmentation point of 6 with a shorter task runtime. Figure 15 shows the comparative experimental results of these two algorithms, and it can be seen from the figure that there is not much difference between the two algorithms in terms of task runtime, and that CECMS is generally about 2% higher than the ADC algorithm in terms of edge memory resource utilization.

For the VGG model, it is found through multiple comparative experiments that the segmentation point calculated by the two algorithms is the same, and both are 10, which is because task runtime is the shortest and edge memory utilization is the highest when the segmentation point is 10.

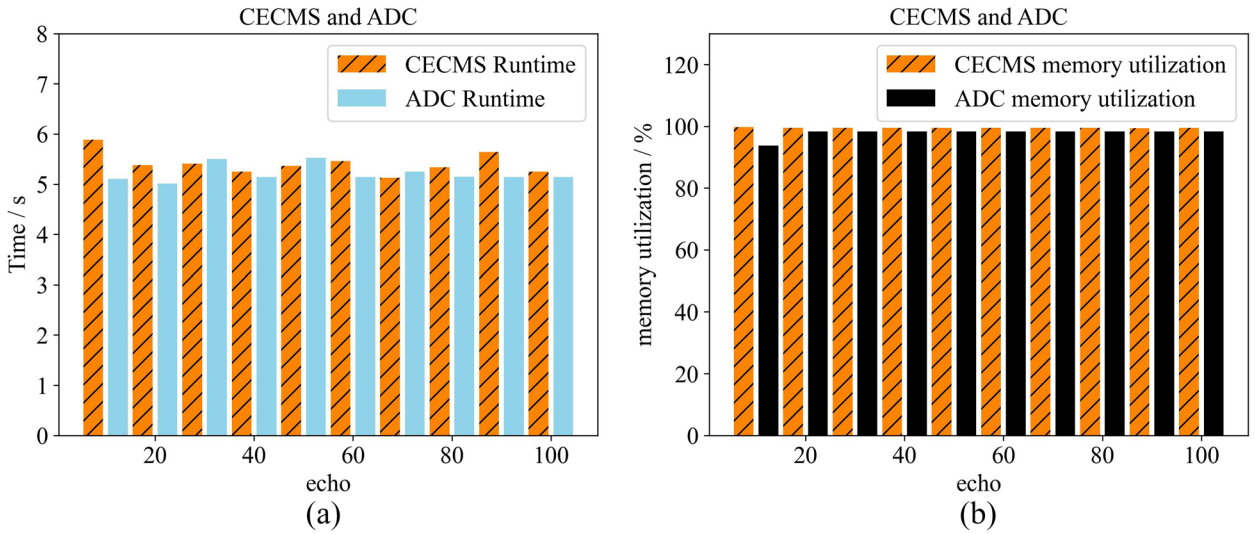

Fig. 15 The Comparison of AlexNet between CECMS and ADC

Based on the aforementioned experiments, a cloud-edge collaborative task scheduling method based on model segmentation effectively reduced the execution time of neural network model tasks in the cloud-edge environment while ensuring the full utilization of edge resources.

Conclusion

This paper proposed a cloud-edge collaborative task scheduling method based on model segmentation to enhance the efficiency of unloading large neural network model tasks in collaborative environments. The proposed method modeled and analyzed the factors influencing the segmentation point of the model, and finally obtained precise factors affecting the calculation of segmentation points via pre-execution. Then, multi-objective solving algorithms were utilized to extract the optimal model segmentation point. For the sake of quick and accurate calculation, the MTMGMO algorithm was designed based on the traditional NSGA-III algorithm. Experimental results indicate that the MTMGMO algorithm is able to improve the calculation speed by 60% compared to single-threaded multi-group computation. Subsequently, tasks were sent to the edge and cloud for task offloading based on the optimal segmentation point. Finally, the effectiveness of the proposed method was verified via simulation and comparative experiments. The proposed method significantly reduces the execution time of neural network model tasks in a cloud-edge collaborative environment while enhancing memory resource utilization of edge nodes, reducing the cost and time required for unloading large neural network model.

Acknowledgements
Not applicable.

Authors′ contributions
Jing Chen as the corresponding author is a major contributor in proposing the method. Chuanfu Zhang implements this method and drafts this manuscript. Wen Li and Yudong Geng carried out the partial experimental work. Hao Sun contributed to the partial data analysis. Tianxiang Zhang, Mingchao Ji and Tonglin Fu contributed to some image processing.

Funding
This work was supported by Project of Key R&D Program of Shandong Province (2022CXGC020106), Qilu University of Technology (Shandong Academy of Sciences) pilot major innovation project of integrating science, education and industry(2022JBZ01-01), Shandong Innovation Ability Improvement Project of Science and Technology small and medium-sized enterprises (2022TSGC1064, 2022TSGC2186), China.

Availability of data and materials
The datasets used during the current study are available from the corresponding author on reasonable request.

Declarations

Competing interests
The authors declare no competing interests.

Received: 27 May 2023 Accepted: 13 March 2024
Published online: 05 April 2024

References

1. Lone AN, Mustajab S, Alam M (2023) A comprehensive study on cybersecurity challenges and opportunities in the IoT world. Security and Privacy 6(6):e318
2. Zhen C, Lin Z, Wang X et al (2023) Cloud–edge collaboration task scheduling in cloud manufacturing: An attention-based deep reinforcement learning approach. Comput In Eng 177:109053
3. Alam M, Shahid M, Mustajab S (2023) Security prioritized multiple workflow allocation model under precedence constraints in cloud computing environment. Cluster Comput 2023:1–36
4. Yadav M, Mishra A (2023) An enhanced ordinal optimization with lower scheduling overhead based novel approach for task scheduling in cloud computing environment. J Cloud Comput 12(1):14
5. Sudheer M, Ganesh RK, Utku K (2023) Multi objective trust aware task scheduling algorithm in cloud computing using whale optimization. J King Saud University 35(2):791–809
6. Kushwaha U, Gupta P, Airen S, et al (2022) Analysis of CNN Model with Traditional Approach and Cloud AI based Approach, 2022 International Conference on Automation. Comput Renewable Syst (ICACRS) 835–842
7. He XY, Qi G, Zhu Z, et al (2023) Medical image segmentation method based on multi-feature interaction and fusion over cloud computing. Simul Model Pract Theory 2023(126):102769
8. Xu H, Zuo L, Sun F, et al (2022) Low-latency Patient Monitoring Service for Cloud Computing Based Healthcare System by Applying Reinforcement Learning. 2022 IEEE 8th Int Conf Comput Commun (ICCC) 2022:1373–1377
9. Hatem K, Mohammed L, Mohammed L et al (2021) Edge Computing Assisted Autonomous Driving Using Artificial Intelligence. Int Wireless Commun Mobile Comput (IWCMC) 2021:254–259
10. Mukherjee M M, Vikas K, Maity D et al (2020) Delay-sensitive and priority-aware task offloading for edge computing-assisted healthcare services. GLOBECOM 2020–2020 IEEE Glob Commun Conf 2020:1–5
11. Satyanarayanan M (2017) The Emergence of Edge Computing. Comput 50(1):30–39
12. Zhang J, Letaief KB (2020) Mobile edge intelligence and computing for the Internet of Vehicles. Proc IEEE 108(2):246–261
13. Fan C, Lu Y, Leng X et al (2020) Data classification processing method for the Power IoT based on cloud-edge collaborative architecture. 2020 IEEE 9th Joint Int Inf Technol Artif Intell Conf (ITAIC) 9:684–687
14. Yang H, Zhao X, Yao Q et al (2022) Accurate fault location using deep neural evolution network in cloud data center interconnection. IEEE Trans Cloud Comput 10(2):1402–1412
15. Chen M, Guo S, Liu K et al (2021) Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing. IEEE Trans Mobile Comput 20(5):2025–2040
16. Eshratifar AE, Abrishami MS, Pedram M (2021) JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services. IEEE Trans Mobile Comput 20(2):565–576
17. Huang Y, Qiao X, Dustdar S et al (2022) Toward decentralized and collaborative deep learning inference for intelligent IoT devices. IEEE Netw 36(1):59–68
18. Kang Y, Hauswald J, Gao C et al (2017) Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. ACM SIGARCH Comput Arch News 45(1):615–629
19. Kum S, Kim Y, Moon J (2019) Deploying Deep Neural Network on Edge-Cloud environment. Int Conf Inf CommunTechnol Convergence (ICTC) 2019:242–244
20. Zhang W, Wang N, Li L et al (2022) Joint compressing and partitioning of CNNs for fast edge-cloud collaborative intelligence for IoT. J Syst Arch 125:102461
21. Hu C, Bao W, Wang D, et al (2019) Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge. IEEE Conf Comput Commun 2019:1423–1431
22. Mehta R, Shorey R (2020) DeepSplit: Dynamic Splitting of Collaborative Edge-Cloud Convolutional Neural Networks. Int Conf Commu Syst Netw (COMSNETS) 2020:720–725

23. Yang S, Zhang Z, Zhao C et al (2022) CNNPC: End-Edge-Cloud Collaborative CNN Inference With Joint Model Partition Compression. IEEE Trans Parallel Distributed Syst. 33(12):4039–4056
24. Gao Z, Miao D, Zhao L et al (2021) Triple-partition Network: Collaborative Neural Network based on the 'End Device-Edge-Cloud.' IEEE Wireless Commun Netw Conf (WCNC) 2021:1–7
25. Xue M, Wu H, Peng G et al (2022) DDPQN: An Efficient DNN Offloading Strategy in Local-Edge-Cloud Collaborative Environments. IEEE Trans Serv Comput 15(2):640–655
26. Zhou L, Wen H, Teodorescu R, et al (2019) Distributing deep neural networks with containerized partitions at the edge. The 10th USENIX Annu Tech Conf, vol 2019. pp 1–7
27. Dey S, Mukherjee A, Pal A, et al (2018) Partitioning of CNN Models for Execution on Fog Devices. 1st ACM Int Workshop 2018:19–24
28. Qararyah F, Wahib M, Dikbayır D et al (2021) A computational-graph partitioning method for training memory-constrained DNNs. Parallel comput 04:102792
29. Teerapittayanon S, Mcdanel B, Kung HT (2017) Distributed Deep Neural Networks over the Cloud, the Edge and End Devices. 2017 IEEE 37th Int Conf Distributed Comput Syst (ICDCS) 2017:328–339
30. Mao J, Yang Z, Wei W et al (2017) MeDNN: A distributed mobile system with enhanced partition and deployment for large-scale DNNs. IEEE/ACM Int Conf Comput-Aided Des (ICCAD) 2017:751–756
31. Ao Y, Wu Z, Yu D, et al (2021) End-to-end Adaptive Distributed Training on PaddlePaddle. arXiv 2021(abs/2112.02752):1–16
32. Hou X, Guan Y et al (2022) Distredge: Speeding up convolutional neural network inference on distributed edge devices. 2022 IEEE Int Parallel Distributed Process Symp (IPDPS) 2022:1097–1107
33. Jeong J, Yang H (2021) Optimal Partitioning of Distributed Neural Networks for Various Communication Environments. Int Conf Artif Intell Inf Commun (ICAIIC) 2021:269–272
34. Miao W, Zeng Z, Wei L, et al (2020) Adaptive DNN Partition in Edge Computing Environments. 2020 IEEE 26th Int Conf Parallel Distributed Syst (ICPADS) 2020:685–690
35. Liu H, Zheng W, Li L, et al (2022) LoADPart: Load-Aware Dynamic Partition of Deep Neural Networks for Edge Offloading. 2022 IEEE 42nd Int Conf Distributed Comput Syst (ICDCS) 2022:481–491
36. He W, Guo S, Guo S et al (2020) Joint DNN Partition Deployment and Resource Allocation for Delay-Sensitive Deep Learning Inference in IoT. IEEE Internet Things J 7(10):9241–9254
37. Zeng J, Liang Z, Zhang J, et al (2022) Research on cloud side collaboration under Internet of vehicles. 2022 IEEE 6th Adv Inf Technol, Electron Automation Control Conf (IAEAC) 2022:245–248
38. Zhang X, Xi Z, Wang T, et al (2022) Source grid load and energy storage management method based on cloud edge cooperation. 2022 7th Asia Conf Power Electrical Eng (ACPEE) 2022:164–169
39. Zhang Y, Wang X, He J et al (2020) A Transfer Learning-Based High Impedance Fault Detection Method Under a Cloud-Edge Collaboration Framework. IEEE Access 8:165099–165110
40. Liu G, Fei D, Xu X et al (2023) An adaptive DNN inference acceleration framework with end–edge–cloud collaborative computing. Future Gener Comput Syst 140:422–435
41. Krizhevsky A, Ilya S et al (2012) ImageNet classification with deep convolutional neural networks. Commun ACM 60:84–90
42. Simonyan K, Andrew Z (2015) Very Deep Convolutional Networks for Large-Scale Image Recognition. Int Conf Learn Representat 2015:1–14

Publisher's Note