EasyChair Preprint

№ 7483

EdgeSP: Scalable Multi-Device Parallel DNN Inference on Heterogeneous Edge Clusters

Zhipeng Gao, Shan Sun, Yinghan Zhang, Zijia Mo and Chen Zhao

February 19, 2022

EdgeSP: Scalable Multi-Device Parallel DNN Inference on Heterogeneous Edge Clusters

Zhipeng Gao[1(✉)], Shan Sun[1(✉)], Yinghan Zhang[2], Zijia Mo[1], and Chen Zhao[1]

[1] State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications, Beijing 100876, China
{gaozhipeng, sunshan, mozijia, zc_zhaochen}@bupt.edu.cn
[2] China International Engineering Consulting Corporation,Ltd, Beijing, China
zyh@ciecc.com.cn

Abstract. Edge computing has emerged as a promising line of research for processing large-scale data and providing low-latency services. Unfortunately, deploying deep neural networks (DNNs) on resource-limited edge devices presents unacceptable latency, hindering artificial intelligence from empowering edge devices. Prior solutions attempted to address this issue by offloading workload to the remote cloud. However, the cloud-assisted approach ignores that devices in the edge environment tend to exist as clusters. In this paper, we propose EdgeSP, a scalable multi-device parallel DNN inference framework that maximizes resource utilization of heterogeneous edge device clusters. We design a multiple fused-layer blocks parallelization strategy to reduce inter-device communication during parallel inference. Further, we add early exit branches to DNNs, empowering the device to trade-off latency and accuracy for a variety of sophisticated tasks. Experimental results show that EdgeSP enables inference latency acceleration of 2.3×-3.7× for DNN inference tasks of various scales and outperforms the existing naive parallel inference method. Additionally, EdgeSP can provide high accuracy inference services under various latency requirements.

Keywords: Edge computing · Edge intelligence · Parallel inference · Deep neural networks · Early-exit · Internet of Things.

1 Introduction

Deep neural networks (DNNs) have become indispensable for handling complex tasks in computer vision, natural language processing, and other fields [1]. While DNNs provide intelligent services with high accuracy, they place higher demands on the computing resources of the devices. At the same time, the number of Internet of Things (IoT) devices has grown exponentially in recent years, and edge computing has emerged to cope with the resulting massive amounts of data and tasks. Edge computing aims to provide low-latency services by performing tasks close to the edge of the network where data is generated, such as end devices or edge servers [2]. It is rewarding to equip DNNs on edge devices, enabling the edge to provide a wide range of intelligent services.

Enable resource-limited edge devices to rapidly execute large DNNs to meet the demands of real-time tasks has attracted extensive research. A common approach is for edge devices to perform DNNs inference tasks collaboratively with the edge server or cloud [3, 4]. In addition, neural networks with early exit branches are receiving more and more attention because of their effectiveness and flexibility [5]. This kind of neural network reduces redundant calculations by allowing simple samples to exit from the shallow layer of the network, thereby significantly reducing inference latency. A triple-partition network with multiple exit branches is proposed [6] based on the early exit mechanism, as shown in Fig. 1. However, this server-assisted approach is critically hampered by the quality of the device's network connection to the remote servers. When the network connection degenerates, the DNN inference time also increases sharply. In addition, transferring local data to edge servers or cloud servers may result in privacy disclosure.

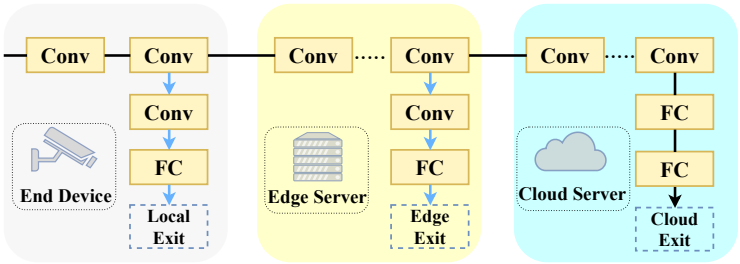

Fig. 1. A sketch of the triple-partition network architecture: simple input samples can be inferred at the DNN branch at the end device, while complex samples require further computation at the edge server and cloud.

Another prospective way to accelerate DNN inference is to perform tasks in parallel by multiple devices, as edge devices typically appear in the form of clusters [7–10]. Some previous work explored the parallel execution of DNN inference on multiple devices [11–13]. However, it is non-trivial to distribute the inference of DNNs on multiple devices. Parallel inference presents data dependency problems since DNNs are inherently tightly coupled [7]. Due to the data dependency problems, existing DNN parallel inference methods incur frequent inter-device communication [14] or substantial overlapping computation [15]. Moreover, none of the above methods can dynamically adjust the inference latency, yet the tasks in edge computing scenarios typically have different requirements. Taking traffic cameras as an example, the task of detecting traffic jams requires low latency but not high accuracy; the task of identifying license plate numbers requires high accuracy, but a moderate amount of latency is acceptable [16]. Therefore, it is indispensable to provide different services according to the application requirements.

To tackle the aforementioned problems, we propose EdgeSP, a scalable multi-device parallel DNN inference framework that leverages the computational resources of heterogeneous edge devices in IoT environments. Multi-device synergy enables fast execution of DNN inference on devices with scarce computing resources. Unlike the server-assisted approaches, EdgeSP resides data on the local trusted devices, avoiding performance instability and privacy disclosure caused by sending data to remote servers. Furthermore, EdgeSP can trade-off between dynamic response time and accuracy to adapt to the needs of different tasks in the IoT environment.

Concretely, our contributions are summarized as follows:

- We propose EdgeSP, a scalable parallel DNN inference framework for heterogeneous edge devices, which accelerates DNN inference and improves the utilization of computational resources in edge clusters.
- We propose a **m**ultiple **f**used-**l**ayer **b**locks (MFLB) parallelization strategy to minimize inter-device communication and overlapping computation overhead, and we further design an adaptive fused-layer workload partition algorithm based on the compute capabilities of heterogeneous devices and dynamic network bandwidth.
- We propose a stepwise method for determining the confidence threshold of exit branches based on task latency requirements so that edge devices can complete the inference task within the specified time.
- We implement EdgeSP on a cluster of heterogeneous edge devices and evaluate its performance on three different scales of DNNs. Experimental results show that EdgeSP is effective in minimizing the inference latency and outperforms prior works.

The rest of this paper is organized as follows. Sect. 2 provides background information. Sect. 3 overviews the design of EdgeSP, followed by the description of technical details. Sect. 4 evaluates the performance of EdgeSP, and Sect. 5 concludes.

2 Related Work

To enable resource-limited edge devices to perform large DNNs, some researchers focus on refining neural network structures to reduce computation. Model pruning is dedicated to removing nonsignificant weights in the DNNs model to reduce calculation [17]. Weight quantization reduces the number of model parameters and calculations by replacing the floating-point number parameters in the original DNN network with low-bit parameters [18]. However, the above methods will degrade the accuracy of the model. The early exit mechanism takes advantage of the variability between samples by adding branches to the shallow layers of the neural network to allow simple samples to complete their inference in advance [5, 19]. Although the early exit mechanism can reduce computation, its acceleration of DNNs inference for edge devices is insufficient [20].

Some researchers are inclined to offload compute-intensive tasks from edge devices to powerful servers. Neurosurgeon [3] proposes to accelerate DNNs inference on edge devices with the help of the cloud. The edge device performs the front part of the DNNs and sends the intermediate data to the cloud, which subsequently performs the rest of the computation. Edgent [21] proposes an end device and edge co-inference method with two early exit points by combining BranchyNet and Neurosurgeon. The Triple-partition Network [6] adds three exit points to traditional DNNs and deploys them to end devices, the edge, and the cloud. However, this server-assisted approach is highly dependent on the quality of the network connection. Worse still, sending local data to edge or cloud servers may result in privacy disclosure.

Another rising star, multi-device parallel inference, has attracted increasing attention. MoDNN [14] first put forward to allocate DNN inference tasks to multiple devices for parallel execution, but its method will cause additional communication overhead. DeCNN [22] reduces frequent inter-device communication during parallel inference by modifying the network structure. Deepthings [15] fuses the first few layers of the DNN network to reduce the communication overhead, which will generate extensive overlapping computations when the number of fused layers is too large. None of the parallel inference architectures mentioned above are scalable, i.e., they cannot provide the flexibility to adjust DNN inference response times according to task requirements.

3 EdgeSP Framework

Our work aims to accelerate DNN inference by leveraging heterogeneous edge device clusters and provide edge devices with the ability to trade-off latency and accuracy to accommodate the varied needs of real-time applications. To do this, we need to address the following issues: (1) how to optimize the impact of data dependency problems caused by parallel inference; (2) how to adaptively distribute tasks to heterogeneous devices in a dynamic network environment; (3) how to empower devices with scalable DNN inference capabilities. Our research focuses on convolutional neural networks (CNNs) as they are widely used in broad-spectrum intelligent services [11].

3.1 Framework Overview

We design a framework, EdgeSP, that can adaptively distribute CNN inference tasks to heterogeneous devices in a dynamic network environment. In order to enable the device to adapt to tasks with different response time requirements, EdgeSP adds several early exit branches to the original CNN network. EdgeSP includes the preparation phase and inference phase. In the preparation phase, we first train CNNs with branching structure and subsequently train the compute capability models of the heterogeneous devices to quantify their performance. As the compute capability of the devices is invariant, each device only needs to be trained once. Each edge device will then broadcast the trained compute

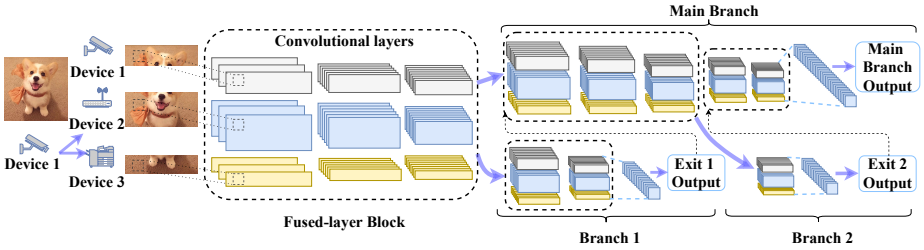

Fig. 2. An example of the inference workflow of EdgeSP. In addition to the main branch, two early exit branches are added to the DNN, namely Branch 1 and Branch 2. An input sample is distributed to three heterogeneous devices, and they execute the DNN in parallel using a multiple fused-layer blocks approach.

capability model to the other devices involved in parallel inference. Eventually, each device will be aware of the compute capabilities of the other devices.

Fig. 2 illustrates an instance of the EdgeSP inference workflow. Three devices execute a CNN with two branches in parallel. The inference workflow is as follows:

- Device **1**, which initiates the CNN inference task, runs the adaptive fused layer workload partition algorithm that assigns different workloads to Device **2** and Device **3** based on network bandwidth and devices' compute capability.
- The three devices perform successive multilayers in a multiple fused layer blocks manner, which will be discussed in Sect. 3.2. The fused layer blocks are followed by synchronization points where the devices will recombine the feature maps.
- When the neural network executes to Branch **1**, each device will save the feature maps computed in the main branch at this point and initiates a new fused layer block. When executing to the fully connected layer, each device sends the computed feature map to the device with the most powerful compute capability in the entire cluster, i.e., Device **2**.
- Then Device **2** will execute the fully connected layer and determine whether to exit the inference in advance at this branch. The details of the early exit mechanism will be elucidated in Sect. 3.4. If Device **2** determines to quit the inference at this point, it will send the result to the task initiating device. Otherwise, Device **2** will send the command to continue execution to Device **1** and Device **3**, and the edge cluster will restore the feature map just retained and continue to execute the main branch.
- When the execution reaches the later branches, the cluster will repeat the above process and determine whether to exit the inference until the whole CNN network is executed.

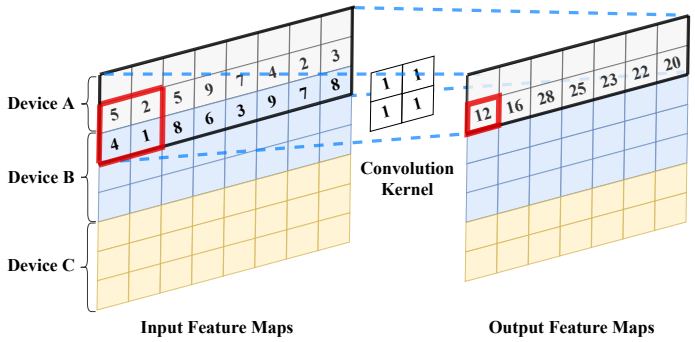

Fig. 3. Schematic diagram of the data dependency problem arising from parallel inference. Device A needs data from device B to compute the output feature map.

3.2 Multiple Fused-Layer Blocks Parallelization Strategy

According to [13], convolutional operations account for more than 70% of the overall execution time of CNN networks, so accelerating the execution of the convolutional layer has become a hot research topic. Since the structure of the convolutional neural network is tightly coupled, distributing the convolution operation to multiple devices incurs data dependency problem. In this section, we present the details of the data dependency problem and the corresponding solutions.

In a convolutional neural network, the convolutional layer extracts massive features from the input samples and passes the results to subsequent convolutional layers to extract higher-level features. For a convolution layer with feature map $\mathbf{M}\{Ch_M, H, W\}$ and convolution kernel $\mathbf{K}\{Ch_K, F, F\}$, the convolution operation can be expressed as follows [23]:

$$\begin{aligned} \mathbf{M} \otimes \mathbf{K} &= \sum_{i=0}^{F-1} \sum_{j=0}^{F-1} \mathbf{M}[Sx+i][Sy+j] \times \mathbf{K}[i][j] \\ 0 \le x < \frac{H-F+S}{S}, &0 \le y < \frac{W-F+S}{S} \end{aligned} \tag{1}$$

where H, W, and Ch_M denote the height, width, and number of input channels of the feature map, F, S, and Ch_K denote the size, stride, and number of output channels of the convolution kernel, respectively.

From Eq. 1, it can be derived that a neuron in the output of the convolution is only relevant to the partial data in the input feature map. This characteristic of the convolution operation provides the possibility of distributing the CNN task to multiple devices for parallel execution. But this characteristic also indicates that the CNN structure is highly coupled, which incurs the data dependency problem. Fig. 3 presents an instance of the data dependency problem. In Fig. 3,

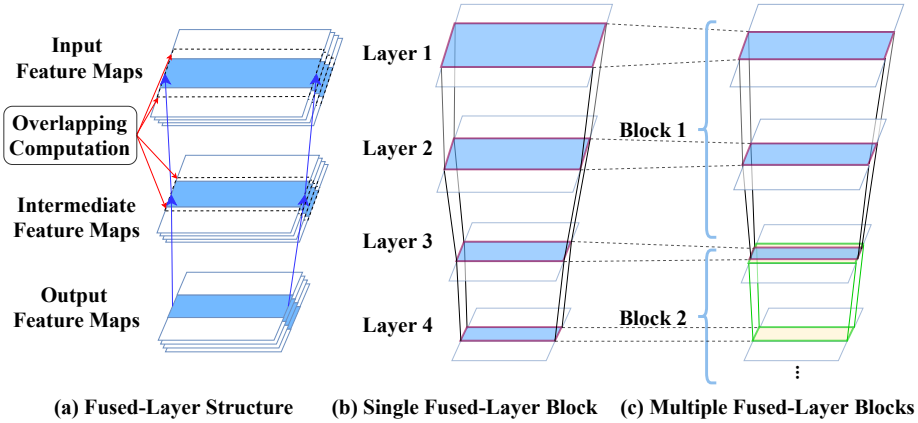

Fig. 4. Comparison of overlap computation caused by fused layer. (a) Schematic diagram of overlap computation caused by fused layer. Variation of computation area size for (b) a single fused layer block and (c) multiple fused layer blocks.

the input feature map is assigned to three devices. Calculating the data in the red box in the output feature map requires the contents of a 2×2 size matrix in the input data, but these data are stored in two different devices. Generally, for a convolution kernel of $F \times F$, each device needs its allocated feature map partition to extend $\lfloor F/2 \rfloor$ along the edges to contain the data required for convolution.

To resolve the data dependency problem, some researchers have adopted a layer-wise approach [11, 12, 14], where each device exchange overlapping data before performing each layer of convolution. This layer-wise approach will undoubtedly incur frequent inter-device communications [13]. We propose a multiple fused-layer blocks (MFLB) parallelization strategy. Each device performs multiple consecutive convolutional layers without exchanging overlapping data during this period, thus avoiding frequent inter-device communication. For workload assignment, we first divide the last layer of the block according to the device compute capability and network bandwidth, and then each workload feature map is extended $\lfloor F/2 \rfloor$ along the edge and recursively to the first layer of the block layer by layer.

As shown in Fig. 4(a), the fused-layer method removes data dependencies by introducing overlapping computations. As the number of fusion layers increases, redundant computations also increase layer by layer. Therefore, we trade off the communication overhead and overlapping computation and adopt multiple fused-layer blocks to reduce the overlapping computation caused by too many fusion layers, as shown in Fig. 4(c). We divide the entire CNN network into blocks, where each device performs the workload within a block consecutively. Each fused block is followed by a synchronization point where each device aggregates and redistributes the feature maps. Fig. 4(b) and Fig. 4(c) are sketches of the naive fused layer method and the MFLB method. In order to calculate Layer

4, the computation amount of the single fused layer block is more than that of multiple fused layer blocks. How to determine the size of each fusion block will be elucidated in Sect. 3.3.

3.3 Workload Partition Algorithms

In this part, we discuss how to distribute workloads to heterogeneous devices adaptively. The fused-layer block assigned to each device can be regarded as a separate task. Each block's last layer is followed by a synchronization point. The goal of distributing workloads is to strive for near-synchronous completion of the fused-layer block tasks by individual devices to avoid long waits at synchronization points.

It was clarified in [24] that the execution time of convolution operation is approximately proportional to the number of floating-point operations (FLOPs) required. To quantify the performance differences between heterogeneous devices $D_k = \{D_1, D_2, \cdots, D_K\}$, in the preparation phase, each device runs a series of convolutional layers with different parameters to train the linear regression model of its compute capability. For a convolutional layer L with feature map $\mathbf{M}\{Ch_M, H, W\}$ and convolutional kernel $\mathbf{K}\{Ch_K, F, F\}$, the FLOPs required can be expressed as follows [25]:

$$FLOPs = 2HW\left(Ch_M F^2 + 1\right) Ch_K \tag{2}$$

The linear regression model of the compute capability of the device D_k is denoted as C_k. Then the execution time for the device D_k to run convolutional layer L can be predicted, which is $C_k(L)$.

Data transmission delay is another factor that affects the execution time of each individual fused-layer block. Edge devices are typically under the same network, so we focus on edge clusters under the same LAN in this work. We use B to denote the network bandwidth. We take a bottom-up approach to analyze the size of the communication data, i.e., we first calculate the size of the last layer of the fused layer block. For device D_k, assuming that the size of the feature map matrix for the last layer of the block is W_{end}, the amount of data required to transmit the last layer is $4W_{end}$ bytes since the size of the floating-point number is 4 bytes. The size of the first layer of the fused layer block, denoted as W_{first}, can be obtained by expanding $\lfloor F/2 \rfloor$ layer by layer along the edge of the W_{end} and recursively to the first layer. Then the total time for device D_k to execute a fused layer block with N layers can be expressed as follows:

$$T_k = \sum_{i=1}^{N} C_k(L_i) + \frac{4\left(W_{first} + W_{end}\right)}{B} \tag{3}$$

We perform one-dimensional workload partitioning of the feature map because one-dimensional partitioning has better performance than two-dimensional partitioning [8].

We propose an adaptive fused-layer workload partition algorithm, as shown in Algorithm 1. This algorithm continuously fine-tunes each device's workload

Algorithm 1 Adaptive Fused-Layer Workload Partition Algorithm

Input:
$\{D_k|k = 1, ..., K\}$: K available devices
$\{C_k|k = 1, ..., K\}$: computation capabilities of K devices
$\{L_{start}, L_{end}\}$: the start and end layer of the block
ζ: waiting time factor
Output:
$\mathcal{S}\{W_k\}$: workload partition strategy

Procedure
for $D_k\,(k = 1, ..., M)$ **do**
  $W_{k_{end}} \leftarrow L_{end} \times \frac{C_k}{\sum_{k=1}^{M} C_k}$, $W_{k_{first}} \leftarrow W_{k_{end}}$
  compute T_k from Eq. 3
end for
$T_{avg} \leftarrow \frac{1}{M} \times \sum_{k=1}^{M} T_k$
$T_k^{diff} \leftarrow abs\,(T_{avg} - T_k)$
if $max\ T_k^{diff} > \zeta \cdot T_{avg}$ **then**
  $W_{argmin(T_k)}$ expands by one pixel
  $W_{argmax(T_k)}$ decreases by one pixel
  Goto Step 6
else
  return $\mathcal{S}\{W_k\}$
end if

based on its compute capability and network bandwidth until the execution time of each device differs by no more than ζ, where ζ is a hyperparameter that adjusts the tolerable wait time of the synchronization point. For example, when ζ is set to 10%, the maximum tolerable wait time at the sync point is 10% of the total execution time of the current fusion layer block.

With the workload partition strategy $\mathcal{S}$, we can further determine the size of each fused-layer block. As mentioned above, the MFLB parallelization strategy reduces inter-device communication overhead while also introducing overlapping computation. The amount of overlapping computation W_{oc} can be obtained by extending $\lfloor F/2 \rfloor$ layer by layer along W_k in $\mathcal{S}$. We design a multiple fused-layer blocks strategy search algorithm that greedily expands the fused-layer block layer by layer until the maximum overlap computation delay is greater than the reduced communication delay or an early exit branch is encountered, as shown in Algorithm 2. This greedy algorithm can select the size of each fused-layer block based on the CNN structure and network bandwidth to minimize the total latency.

3.4 Early Exit Mechanism

To meet the needs of different real-time tasks, we leverage the early exit mechanism to provide devices with the ability to trade-off latency and accuracy. The early exit mechanism adds branches to the original CNN, allowing simple input samples to be inferred in a shallow layer of the CNN. Here entropy is used to

Algorithm 2 Multiple Fused-Layer Blocks Strategy Search Algorithm

Input:
$\{C_k|k=1,...,K\}$: computation capabilities of K devices
$\{L_i|i=1,...,N\}$: CNN model with N layers
E: set of early exit branches
B: network bandwidth
W_{oc}: overlapping feature map
Output:
$\mathcal{F}$: multi fused layer strategy

Procedure
$L_{start} \leftarrow L_1$, $L_{end} \leftarrow L_{start}$
if $L_{end+1} \neq L_N$ **and** $L_{end+1} \notin E$ **then**
  $L_{end} \leftarrow L_{end+1}$
  execute Algorithm 1 with $\{L_{start}, L_{end}\}$
  if $max\ C_k\left(W_{oc}\right) > max \sum_{i=first+1}^{end-1} 4Wi/B$ **then**
    add $\{L_{start}, L_{end}\}$ *to* $\mathcal{F}$
    $L_{start} \leftarrow L_{end+1}, L_{end} \leftarrow L_{start}$
  end if
  goto Step 3
else if $L_{end+1} \in E$ **then**
  goto Step 7
else
  add $\{L_{start}, L_{end}\}$ *to* $\mathcal{F}$
end if
return $\mathcal{F}$

evaluate how confident the branch is about the input sample. Entropy is defined as:

$$entropy\left(y\right) = \sum_{c \in \mathcal{C}} y_c \log y_c \tag{4}$$

where y is a vector containing computed probabilities for all possible class labels and $\mathcal{C}$ is a set of all possible labels [5]. It is worth noting that EdgeSP has multiple synchronization points, which are well adapted to the added branches.

The confidence threshold for each exit branch needs to be dynamically scaled according to the task latency requirements. For each branch $n \in \{1, 2, \cdots, N\}$, the probability that a sample exits at this branch is P_n ($P_n \in [0, 1]$), where N is the main branch. We can predict the execution time T_n for each branch based on Eq. 3. Then for a given task time threshold T_{th}, P_n should satisfy the following constraints:

$$\sum_{n=1}^{N} P_n \times T_n \leq T_{th}, \ \sum_{n=1}^{N} P_n = 1 \tag{5}$$

We propose a stepwise method for determining branches confidence thresholds with the following procedure:

- In the training phase, we record a list of entropy values for the entire training set samples at each exit branch, denoted as $\mathcal{L}_n$
- In the inference phase, a set of eligible P_n values is generated based on the delay requirement T_{th}, and the set with the highest percentage of posterior exit branches is selected to obtain higher accuracy.
- The entropy value at P_1 of the entropy list $\mathcal{L}_1$ is chosen as the confidence threshold of the first branch.
- Subsequently, the first $\sum_{i=1}^{n-1} P_i$ values are eliminated from the entropy list $\mathcal{L}_n$, and then we choose the entropy value at $\frac{P_n}{1-\sum_{i=1}^{n-1} P_i}$ as the confidence threshold for exit branch n.

We note that adding too many exit branches is inadvisable since complex samples need to go through each branch without being able to exit inference earlier. Therefore, increasing the number of branches, while providing a more fine-grained service, also leads to an increase in the average inference latency. The exit branch at the shallow level of the CNN fails to give high confidence results due to the insufficient features extracted. Therefore the exit branch should not be positioned too close to the front of the models. Previous work has demonstrated that branches added at 1/2 and 3/4 of the CNNs can achieve the satisfactory speedup without excessive loss of accuracy [26]. In this work, for comparison with the server-assisted architecture, we added exit branches at 1/2 and 3/4 of the original CNNs to simulate the exit branches at the edge server and the cloud, respectively.

4 Evaluation

We implement EdgeSP in a cluster of heterogeneous edge devices and evaluate its performance under different device counts and network bandwidths. Moreover, we further test the average inference accuracy of EdgeSP under different latency requirements.

Table 1. Heterogeneous Edge Devices Used in Experiments

Device	CPU Frequency	Memory
Raspberry Pi 4B × 2	1.5 GHz	4 GB
Virtual Machine × 2	1000 MHz	4 GB
Virtual Machine × 2	800 MHz	4 GB

4.1 Experiment Settings

We simulate edge device clusters with heterogeneous computing capabilities with the devices in Table 1. We increase the number of devices in the edge cluster

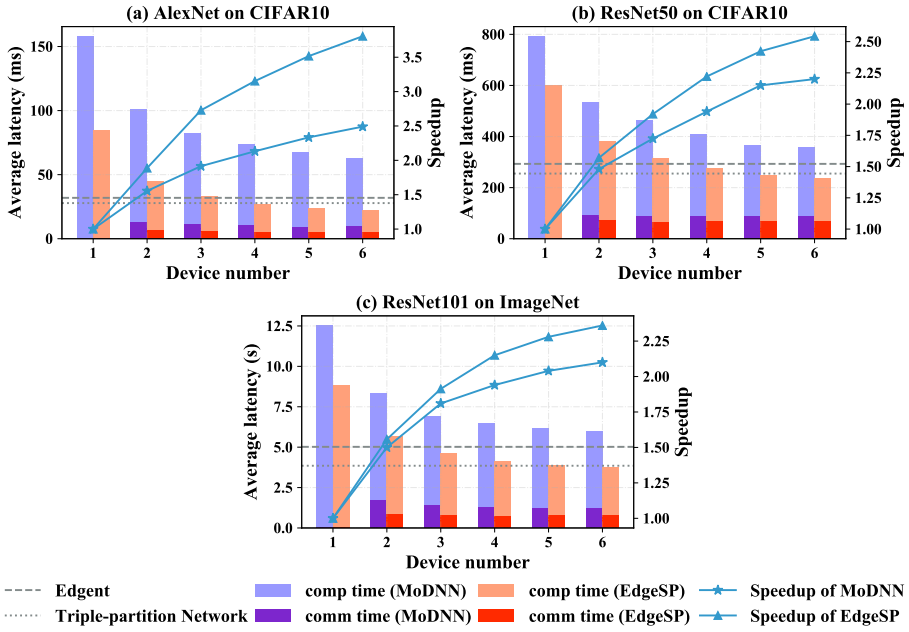

Fig. 5. The performance of EdgeSP at different device counts. Its average latency is compared with MoDNN, Edgent, and Triple-partition Network.

from 1 to 6 in the order shown in Table 1. For comparison with the Edgent [21] and Triple-partition Network [6], we use a PC with an i5-8400 CPU to simulate the edge server and a server with four GTX3090 GPUs to simulate the cloud. We implement EdgeSP on AlexNet [27], ResNet50 [1], and ResNet101 [1] as they represent CNN models with different depths. The CIFAR10 [28] and ImageNet [29] datasets are employed to evaluate the performance of EdgeSP in tasks of varying difficulty. We use the WonderShaper [21] tool to adjust the available bandwidth between devices.

4.2 Performance Comparison

The variation of the average inference delay for each CNN model with an exit rate of P_n = [40%, 40%, 20%] and network bandwidth of 100 Mbps are shown in Fig. 5. It's a representative case as the entire inference workflow described in Sect. 3.1 is executed at this exit rate, and different CNN models can achieve satisfactory acceleration performance. Our framework achieves desired performance in CNN inference tasks of three different sizes and difficulties. We can observe that the average latency of CNN tasks decreases as the number of devices increases. Benefiting from the early exit mechanism, the average latency of EdgeSP is significantly lower than MoDNN in all three CNN tasks. Moreover,

a lower communication latency than MoDNN is achieved thanks to the MFBL parallelization strategy. When the number of devices exceeds five, EdgeSP can complete the inference task faster than Edgent and Triple-partition Network. Although they speed up the inference with the help of edge servers and the cloud, tasks that reside on end devices cannot be accelerated. Furthermore, EdgeSP does not involve uploading data to third-party servers, thus avoiding the risk of privacy disclosure.

As the number of devices increases, the acceleration ratio curves of EdgeSP and MoDNN flatten out, but EdgeSP consistently achieves a higher acceleration ratio than MoDNN. This trend is due to the increase in overlapping computation and communication overhead, which suggests that involving too many devices in parallel inference is not justifiable. Previous work [13] has demonstrated that when the number of devices exceeds six, the additional overhead significantly diminishes the acceleration effect.

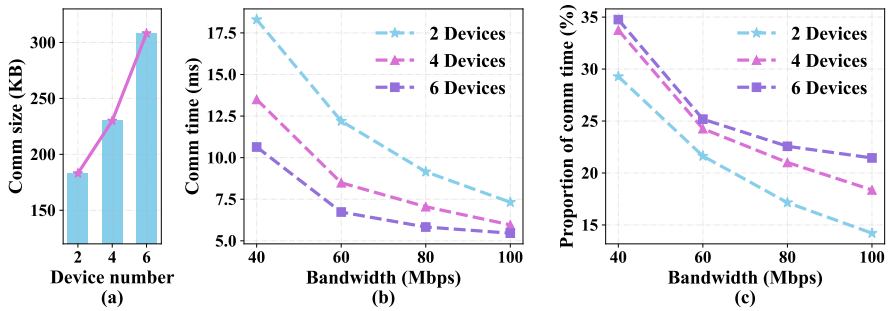

Fig. 6. AlexNet communication overhead at different bandwidths. (a) Variation of total communication data size with the number of devices. (b) Variation of communication time with bandwidth. (c) Variation of communication time as a percentage of total time with bandwidth.

4.3 Analysis of the Communication Overhead

Fig. 6 shows the impact of network bandwidth on the performance of EdgeSP using AlexNet as an example. As the number of devices increases in Fig. 6(a), the workload is divided into smaller areas, but the overlap of tasks between devices increases. Therefore, the amount of data to be transferred also increase. The smaller the number of devices, the smaller the total communication size, but the more workload is allocated to each device. Therefore, the communication time is higher with fewer devices, as shown in Fig. 6(b). Combining Fig. 6(b) and Fig. 6(c), we can see that the communication time gradually decreases as the network bandwidth increases. The more devices there are, the faster the computing task will be completed, so the communication time occupies a higher

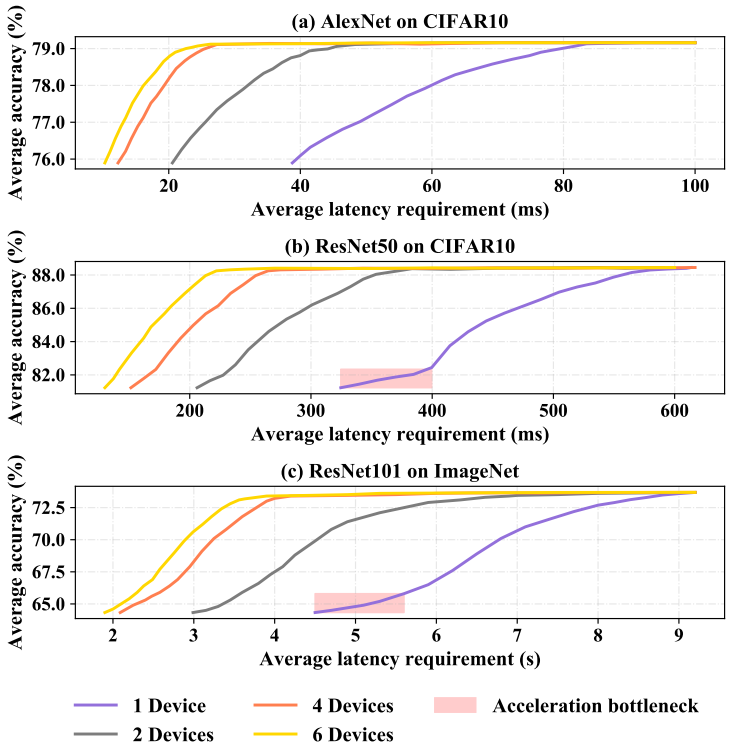

Fig. 7. Variation of the average accuracy of the three CNNs for different task latency requirements.

percentage when there are more devices. On the other hand, multi-device can complete the task faster, which means that EdgeSP can mitigate the impact of bandwidth reduction to some extent.

4.4 Performance under Different Latency Requirements

EdgeSP is capable of adjusting the inference time according to the task latency requirements. Taking AlexNet in Fig. 7(a) as an example, when the latency requirement is 59ms, the inference accuracy of a single device can only reach 77.9%, while the inference accuracy of multiple devices can reach 79.2%, and a single device cannot complete the task within 38ms. As the number of devices increases, EdgeSP can achieve higher accuracy with a specified latency requirement. For example, when the latency requirement is 24ms, the inference accuracy of two devices is 76.6%, while the inference accuracy of six devices can be as high as 79%. Due to the complexity of ImageNet, the accuracy of early exit branches is not as high as on CIFAR10, so the average accuracy in Fig. 7(c)

will have more attenuation than in Fig. 7(b), but is still within an acceptable range. In Fig. 7(b) and Fig. 7(c), ResNet with multiple exit points running on a single device suffers from the acceleration bottleneck phenomenon due to its complex architecture. The acceleration bottleneck arises because the accuracy of the preceding exit branch is unsatisfactory, and the later branches require a longer execution time. Most input samples can only exit from the preceding branches with lower accuracy to meet the latency requirement. However, multi-device parallelism effectively suppresses the impact of acceleration bottlenecks. In a word, the multi-device parallelism and early exit mechanism complement each other, thus enabling EdgeSP to provide scalable and fast DNN inference capability for edge devices.

5 Conclusion

In this paper, we propose EdgeSP, a scalable multi-device parallel DNN inference framework, which substantially reduces the latency of DNN execution by resource-limited edge devices. We design a multiple fused-layer blocks parallelization strategy to minimize the communication overhead incurred by parallel inference. In addition, we add early exit branches to the original DNNs and propose a stepwise confidence threshold determination method, which empowers the device to trade-off latency and accuracy. Experimental evaluations show that EdgeSP achieves lower latency than server-assisted approaches and naive parallel inference in tasks of different scales. Furthermore, EdgeSP enables scalable inference for edge devices to provide high-accuracy services under various latency requirements. In future work, we plan to explore multi-device parallel execution of fully connected layers to accelerate DNN inference further.

Acknowledgments

This work was supported in part by the General Program of National Natural Science Foundation of China under Grant 62072049, and in part by the National Key Research and Development Project of China under Grant 2019YFB2103202 and Grant 2019YFB2103200.

References

1. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90
2. Xiao, K., Gao, Z., Shi, W., Qiu, X., Yang, Y., Rui, L.: Edgeabc: An architecture for task offloading and resource allocation in the internet of things. Future Gener. Comput. Syst. **107**, 498–508 (2020). https://doi.org/10.1016/j.future.2020.02.026
3. Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., Tang, L.: Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. SIGARCH Comput. Archit. News **45**(1), 615–629 (Apr 2017). https://doi.org/10.1145/3093337.3037698

4. Teerapittayanon, S., McDanel, B., Kung, H.: Distributed deep neural networks over the cloud, the edge and end devices. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 328–339 (2017). https://doi.org/10.1109/ICDCS.2017.226
5. Teerapittayanon, S., McDanel, B., Kung, H.T.: Branchynet: Fast inference via early exiting from deep neural networks. CoRR **abs/1709.01686** (2017), http://arxiv.org/abs/1709.01686
6. Gao, Z., Miao, D., Zhao, L., Mo, Z., Qi, G., Yan, L.: Triple-partition network: Collaborative neural network based on the 'end device-edge-cloud'. In: 2021 IEEE Wireless Communications and Networking Conference (WCNC). pp. 1–7 (2021). https://doi.org/10.1109/WCNC49053.2021.9417243
7. Du, J., Shen, M., Du, Y.: A distributed in-situ cnn inference system for iot applications. In: 2020 IEEE 38th International Conference on Computer Design (ICCD). pp. 279–287 (2020). https://doi.org/10.1109/ICCD50377.2020.00055
8. Zhang, S., Zhang, S., Qian, Z., Wu, J., Jin, Y., Lu, S.: Deepslicing: collaborative and adaptive cnn inference with low latency. IEEE Transactions on Parallel and Distributed Systems **32**(9), 2175–2187 (2021)
9. Mohammed, T., Joe-Wong, C., Babbar, R., Francesco, M.D.: Distributed inference acceleration with adaptive dnn partitioning and offloading. In: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications. pp. 854–863 (2020). https://doi.org/10.1109/INFOCOM41043.2020.9155237
10. Xue, F., Fang, W., Xu, W., Wang, Q., Ma, X., Ding, Y.: Edgeld: Locally distributed deep learning inference on edge device clusters. In: 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). pp. 613–619 (2020). https://doi.org/10.1109/HPCC-SmartCity-DSS50907.2020.00078
11. Zeng, L., Chen, X., Zhou, Z., Yang, L., Zhang, J.: Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices. IEEE/ACM Transactions on Networking **29**(2), 595–608 (2021). https://doi.org/10.1109/TNET.2020.3042320
12. Mao, J., Yang, Z., Wen, W., Wu, C., Song, L., Nixon, K.W., Chen, X., Li, H., Chen, Y.: Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns. In: 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp. 751–756. IEEE (2017)
13. Zhou, L., Samavatian, M.H., Bacha, A., Majumdar, S., Teodorescu, R.: Adaptive parallel execution of deep neural networks on heterogeneous edge devices. In: Proceedings of the 4th ACM/IEEE Symposium on Edge Computing. p. 195–208. SEC '19, Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3318216.3363312
14. Mao, J., Chen, X., Nixon, K.W., Krieger, C., Chen, Y.: Modnn: Local distributed mobile computing system for deep neural network. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2017. pp. 1396–1401 (2017). https://doi.org/10.23919/DATE.2017.7927211
15. Zhao, Z., Barijough, K.M., Gerstlauer, A.: 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), 2348–2359 (2018)
16. Fang, B., Zeng, X., Zhang, M.: Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In: Proceedings of the 24th

Annual International Conference on Mobile Computing and Networking. p. 115–127. MobiCom '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3241539.3241559, https://doi.org/10.1145/3241539.3241559
17. Xu, Z., Yut, F., Liu, C., Chen, X.: Reform: Static and dynamic resource-aware dnn reconfiguration framework for mobile device. In: 2019 56th ACM/IEEE Design Automation Conference (DAC). pp. 1–6 (2019)
18. Oh, Y.H., Quan, Q., Kim, D., Kim, S., Heo, J., Jung, S., Jang, J., Lee, J.W.: A portable, automatic data qantizer for deep neural networks. In: Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques. PACT '18, Association for Computing Machinery, New York, NY, USA (2018), https://doi.org/10.1145/3243176.3243180
19. Tan, X., Li, H., Wang, L., Huang, X., Xu, Z.: Empowering adaptive early-exit inference with latency awareness. Proceedings of the AAAI Conference on Artificial Intelligence **35**(11), 9825–9833 (May 2021), https://ojs.aaai.org/index.php/AAAI/article/view/17181
20. Laskaridis, S., Venieris, S.I., Almeida, M., Leontiadis, I., Lane, N.D.: Spinn: Synergistic progressive inference of neural networks over device and cloud. In: Proceedings of the 26th Annual International Conference on Mobile Computing and Networking. MobiCom '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372224.3419194
21. Li, E., Zeng, L., Zhou, Z., Chen, X.: Edge ai: On-demand accelerating deep neural network inference via edge computing. IEEE Transactions on Wireless Communications **19**(1), 447–457 (2020). https://doi.org/10.1109/TWC.2019.2946140
22. Du, J., Zhu, X., Shen, M., Du, Y., Lu, Y., Xiao, N., Liao, X.: Model parallelism optimization for distributed inference via decoupled cnn structure. IEEE Transactions on Parallel and Distributed Systems **32**(7), 1665–1676 (2021). https://doi.org/10.1109/TPDS.2020.3041474
23. Zhao, K., Di, S., Li, S., Liang, X., Zhai, Y., Chen, J., Ouyang, K., Cappello, F., Chen, Z.: FT-CNN: algorithm-based fault tolerance for convolutional neural networks. IEEE Trans. Parallel Distributed Syst. **32**(7), 1677–1689 (2021). https://doi.org/10.1109/TPDS.2020.3043449, https://doi.org/10.1109/TPDS.2020.3043449
24. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European conference on computer vision (ECCV). pp. 116–131 (2018)
25. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient transfer learning. arXiv preprint arXiv:1611.06440 **3** (2016)
26. Zhang, L., Tan, Z., Song, J., Chen, J., Bao, C., Ma, K.: Scan: A scalable neural networks framework towards compact and efficient models. arXiv preprint arXiv:1906.03951 (2019)
27. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25**, 1097–1105 (2012)
28. Krizhevsky, A.: Learning multiple layers of features from tiny images pp. 32–33 (2009), https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
29. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 248–255 (2009). https://doi.org/10.1109/CVPR.2009.5206848