EdgeLD: Locally Distributed Deep Learning Inference on Edge Device Clusters

Feng Xue*, Weiwei Fang*†, Wenyuan Xu*, Qi Wang*, Xiaodong Ma*, Yi Ding‡
*School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China
†Key Laboratory of Industrial Internet of Things & Networked Control, Ministry of Education, Chongqing, China
‡School of Information, Beijing Wuzi University, Beijing, China
*{17120431, fangww, 19120419}@bjtu.edu.cn; *{2500282065, 1054588756}@qq.com; ‡dingyi@bwu.edu.cn

***Abstract*—Deep Neural Networks (DNN) have been widely used in a large number of application scenarios. However, DNN models are generally both computation-intensive and memory-intensive, thus difficult to be deployed on resource-constrained edge devices. Most previous studies focus on local model compression or remote cloud offloading, but overlook the potential benefits brought by distributed DNN execution on multiple edge devices. In this paper, we propose EdgeLD, a new framework for locally distributed execution of DNN-based inference tasks on a cluster of edge devices. In EdgeLD, DNN models' time cost will be firstly profiled in terms of computing capability and network bandwidth. Guided by profiling, an efficient model partition scheme is designed in EdgeLD to balance the assigned workload and the inference runtime among different edge devices. We also propose to employ layer fusion to reduce communication overheads on exchanging intermediate data among devices. Experiment results show that our proposed partition scheme saves up to 15.82% of inference time with regard to the conventional solution. Besides, applying layer fusion can speedup the DNN inference by 1.11-1.13X. When combined, EdgeLD can accelerate the original inference time by 1.77-3.57X on a cluster of 2-4 edge devices.**

***Index Terms*—deep learning, distributed inference, deep neural network, edge computing.**

I. Introduction

Deep Neural Network (DNN) has become the most important technology that enables a variety of computer vision and natural language processing applications [1]. With the growing need for high accuracy, there is a clear trend to increase the size and complexity of DNN models. However, this incurs heavy computation workload and high resource consumption. For example, AlexNet has 61M weights and requires 724M multiply-and-accumulates (MACs) per image, and VGG16 has 138M weights and requires 15.5G MACs per image [2]. Such resource demands become prohibitive for edge devices, e.g., embedded boards and smartphones, due to limitations in processing capacity and memory space [1]. It is generally intolerable in practice if the on-device DNN model takes a long time duration for obtaining the classification result for one image [3].

This work is partially supported by the Beijing Municipal Natural Science Foundation under Grant L191019, the Open Project of Key Laboratory of Industrial Internet of Things & Networked Control, Ministry of Education under Grant 2019FF03, the Open Project of Beijing Intelligent Logistics System Collaborative Innovation Center under Grant BILSCIC-2019KF-10, and the CERNET Innovation Project under Grant NGII20190308. Weiwei Fang is the corresponding author.

In recent years, researchers have proposed many approaches to close the gap between resource demands of DNN models and resource constraints of edge devices [1]. These studies can be classified into two kinds, namely, model compression and cloud offloading. DNN compression techniques seek to reduce the number of operations and model size, by weight pruning, knowledge distillation or compact architecture design [2], [4]. However, eliminating model redundancies requires complex pre-processing [7], and incurs accuracy loss inevitably. Cloud offloading schemes divide the DNN model and offload the computational-intensive part to a cloud server, so as to alleviate the heavy workload of edge device. Nevertheless, the inference performance is highly dependent on unpredictable network conditions [5]. Furthermore, these schemes may raise concern about data privacy in the offloading process [6].

To address the aforementioned challenges, we propose EdgeLD - a new framework that aims to solve the resource constraint problem by executing inference of unmodified DNN models in a locally distributed manner on a cluster of edge devices. Compared to prior work, EdgeLD can better adapt to the variation of computing resources and network bandwidth [3], [7], and provide protection for data privacy (the cluster can be privately owned) [4], [5]. In summary, we make the following contributions in this paper:

1) We first profile the device's inference time cost in terms of computing capability and network bandwidth. The obtained profiling model provides the decision-making basis for the following model partition.
2) Inspired by the MapReduce paradigm, we propose a DNN model partition scheme for dividing and assigning the workload to heterogeneous edge devices, so as to optimize execution parallelism.
3) We further introduce a layer fusion scheme to reduce communication overheads on exchanging intermediate data among edge devices.

Experiments with EdgeLD show that it can provide scalable speedups for DNN inference under various parameter settings, and significantly outperform the prior work [3]. The EdgeLD codebase is released in open-source form at https://github.com/fangvv/EdgeLD.

The rest of this paper is organized as follows. Section II reviews the related work. Section III describes the proposed

EdgeLD framework in detail. Section IV presents experimental results for performance evaluation. Finally, Section V concludes this paper.

II. Related Work

Researchers have proposed different methods to alleviate the tension between resource-hungry DNN models and resource-constrained edge devices. In the following, we outline several important prior work on model compression, cloud offloading and distributed inference. Due to the space limitation, more detailed introduction can be referred to [1], [2].

A. Model Compression

In order to directly running DNN inference on edge devices, various model compression approaches have been proposed. For example, weight pruning based schemes attempt to remove the redundant and unimportant model parameters of a trained DNN to reduce both computational complexity and model size [4], [8]. The knowledge distillation aims to distill a compact and fast network that performs as well as the large complex network [9], [10]. However, these techniques may suffer from design complexity and accuracy loss [3]. More recently, some compact and efficient DNN models are proposed, by making use of the multiple group convolution strategy and 1×1 convolution [2], [11].

B. Cloud Offloading

To alleviate the unaffordable workload of DNN inference, an intuitive idea is to decouple an DNN model according to the execution sequence, and offload the latter part which is often computing-intensive onto a powerful cloud server. The key challenge here is how to well partition so as to achieve the optimal latency performance. Neurosurgeon [12] proposes a regression model for predicting the layer latency and energy consumption, and selects an optimal partition point that satisfying either the latency requirement or the energy requirement. JALAD [13] formulates the partition problem as an Integer Linear Programming problem to minimize the runtime under the given accuracy requirement. However, such techniques may suffer from excessive long transmission latency due to unstable wide-area network connections [7].

C. Distributed Inference

MoDNN [3] is the first work designed to orchestrate DNN inference among multiple mobile devices in the same WLAN. It proposes a Biased One-Dimensional Partition (BODP) scheme to partition individual layers into strips for parallel execution in a MapReduce fashion. MeDNN [7], as a follow-up work, develops a Greedy Two-Dimensional Partition (GTDP) scheme for layer partitioning. The 2D-like partitioning strategy will significantly increase the overlap parts of data regions of different partitions, resulting in more communication and synchronization overhead. Note that both MeDNN and MoDNN process the DNN model in a layer-by-layer manner. In contrast, DeepThings [6] proposes a Fused Tile Partitioning (FTP) method to partition the fused layers vertically in a grid fashion, so as to reduce memory footprint and communication overhead. However, DeepThings is only applicable for a computing cluster with homogeneous devices. What's more, none of these prior work has taken the impact of network conditions among devices into consideration.

III. EdgeLD Framework

A. Framework Overview

EdgeLD implements MapReduce-like distributed inference on a cluster of edge devices with same or different computation and communication resources. In this cluster, one device is appointed or elected as the group leader (GL), and dedicated for data storage and model partitioning. The other devices act as follower nodes (FNs) to share the inference workload from the GL. All these devices are inter-connected by wired or wireless links.

According to the performance analysis on DNN layers in [7], convolutional layers dominates the computation time and the memory footprint, either of which is at least 10.3 times higher than that of fully-connected layers. Meanwhile, the other types of layers, e.g., pooling or ReLu layers, remains negligible. Thus, EdgeLD chooses to only offload the computation of convolutional layers to FNs, and leave the other types of layers to GL [6], [7].

As shown in Fig. 1, the EdgeLD framework is consisted of the following operations: 1) Profiling: EdgeLD profiles each type of FNs to generate time prediction models for convolution computations and data transmissions; 2) Mapping: Guided by profiling, the GL runs the partition algorithm to the given convolutional layer or layers, and assigns the partitioned part of workload to each corresponding FN; 3) Reducing: After all FNs have finished processing their work, the GL will collect and merge the subparts of output from these devices; 4) Local Computing: Except for the convolutional layers, all the other layers (e.g., fully-connected layer) are computed locally at the GL in the conventional centralized way.

B. Device-Specific Time Profiling

Given an edge device, the inference consumption of a DNN model can be generally evaluated by the execution time. In EdgeLD, the execution time of a FN is consisted of the time on convolution computation and the time on data transmission.

As reveled in [14], given the computation hardware and a DNN model, it can be considered that the computation time is approximately proportional to the number of floating-point operations (FLOPs) required by the neural network layers. For convolutional kernels, the FLOPs F can be given as [15]:

$$F = 2HW(C_{in}K^2 + 1)C_{out}, \tag{1}$$

where H and W denote height and width of the input feature map, K denotes the kernel size, C_{in} and C_{out} denote the number of input and output channels, respectively.

Based on (1), we establish linear regression models [14] about time consumption of convolution computation T_{comp}^{conv} for each hardware type of FN. Specifically, measurement

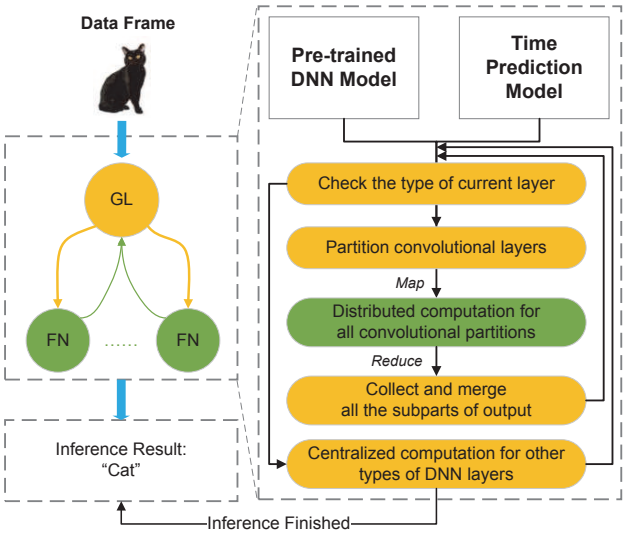

Fig. 1. Overview of the EdgeLD framework.

experiments are carried out on convolution computation with various settings of parameters, including H, W, C_{in} and C_{out}. Suppose we have obtained a set of measurement results $\{(f_1, t_1), (f_2, t_2), \dots (f_n, t_n)\}$, the model can be given based on the least-squares regression technique as:

$$T_{comp}^{conv} = aF + b, \tag{2}$$

where the coefficients a and b are given as follows,

$$a = \frac{n\sum_{i=1}^{n}(f_i t_i) - \sum_{i=1}^{n} f_i \sum_{i=1}^{n} t_i}{n\sum_{i=1}^{n} f_i^2 - (\sum_{i=1}^{n} f_i)^2},$$

$$b = \frac{\sum_{i=1}^{n} t_i - a\sum_{i=1}^{n} f_i}{n}.$$

It can be inferred from (2) that the coefficient a is inversely proportional to the computation capability, denoted by c, of the given device. Thus, in this paper the reciprocal of a is used to measure and represent c, i.e., $c = \frac{1}{a}$.

As the other key factor, the communication time T_{comm}^{conv} is dependent on the transmission amount and the link bandwidth from a FN to the GL. The real-time link bandwidth can be easily measured by the well-known software iPerf.

Based on the analysis above, each FN is able to predict the execution time cost T^{pre} by summing up $(T_{comp}^{conv} + T_{comm}^{conv})$ for the assigned convolution computation workload from one layer or multiple layers. EdgeLD will use the prediction results to improve execution parallelism among all FNs.

C. Convolutional-Layer Computation Partitioning

According to [6], [7], convolutional layers contribute to the majority (e.g., about 90%) of both computation time and memory footprint in DNN inference. An intuitive idea is to partition the original convolutional layer into independently distributable execution parts, so as to enable inference speedup with multiple devices. Similar to BODP in [3], we propose

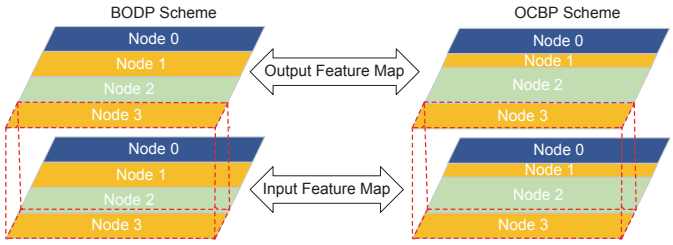

Fig. 2. An illustration of partition results for BODP and OCBP. The cluster is comprised of 4 FNs with the same computation capabilities but different network conditions.

Algorithm 1 OCBP Partition Scheme

Input:
M: the total number of FNs
$\{c_i\}$: the computation capabilities of FNs
W: the width of original feature map
$W_{step} = 1$: the adjusting step size of partition width
$\{T_i^{pre}\}$: the predicted time costs of FNs
$T_{tolerate}$: the tolerable difference of time cost
Output:
$\{W_i\}$: the workload partitions for FNs

1: **Procedure**
2: **for** $i = 1, \dots, M$ **do**
3: Calculate the initial partition width for FN i,
4: $W_i = Round(\frac{c_i}{\sum_{i=1}^{M} c_i} * W)$
5: **end for**
6: **for** $i = 1, \dots, M$ **do**
7: Obtain the required FLOPs F_i based on W_i for FN i
8: Use F_i to predict the time costs $\{T_i^{pre}\}$
9: **end for**
10: Find $T_{max}^{pre} = \max(T_i^{pre})$ and $T_{min}^{pre} = \min(T_i^{pre})$
11: $T_{diff} = T_{max}^{pre} - T_{min}^{pre}$
12: **if** $T_{diff} < T_{tolerate}$ or $\exists W_i = W_{step}, (i = 1, \dots, M)$ **then**
13: **return** the partitioned width $\{W_i\}$ for FNs
14: **else**
15: $W_{\arg\max(T_{max}^{pre})} = W_{\arg\max(T_{max}^{pre})} - W_{step}$
16: $W_{\arg\min(T_{min}^{pre})} = W_{\arg\max(T_{min}^{pre})} + W_{step}$
17: Goto Step 6
18: **end if**

a One-dimensional Computation&Bandwidth-based Partition (OCBP) scheme to partition the convolutional layers. The main difference between these two schemes is that OCBP takes not only computational resources but also network conditions into consideration. The reason is that in a real-world cluster environment, network latency is so important a factor for application performance that can't be simply neglected [16].

The OCBP partition scheme is illustrated in Algorithm 1. We first divide the original layer computation workload, i.e., the input feature map, into multiple strip-like partitions based on computation capabilities obtained in device profiling. Next, we iteratively adjust and balance the workload distributions among the FNs, according to the predictions on execution

time. Specifically, OCBP searches for the two devices with the smallest and largest time consumption, and then adjusts the partition width of them by increasing or decreasing by one step size W_{step}, respectively. At last, if the partition results are acceptable (i.e., $T_{diff} < T_{tolerate}$), or there exists one FN that is extremely inferior to the other ones (i.e., $\exists W_i = W_{step}, (i = 1, \ldots, M)$), OCBP terminates and returns the final partitions for FNs.

D. Inter-Device Data Exchanging

The convolution computation in EdgeLD is executed in a layer-wise parallel manner. Each FN is only responsible for computing a part of output of the current layer, while the GL will collect, merge and re-partition all the subparts of the output before executing the next layer [3], [7]. However, such a layer-wise strategy will bring about considerable communication overhead and memory footprint, resulting in a negative impact on speedup performance.

To avoid frequent data exchanges between FN and GL, we further propose to parallelize multiple consecutive layers as a single fused block [6]. Specifically, all convolutional layers in a fused block are computed layer-by-layer by the same FN, and the FN communicates with GL only for receiving the input of the first layer and sending the output of the last layer. In this way, the frequency of data exchanges and the corresponding communication costs can be well reduced. Note that a fused block merely contains several consecutive convolutional layers, and should not contain any other type of DNN layers (e.g., fully-connected layer).

Due to the nature of convolution computation [2], the output of a layer partition depends on not only the corresponding data region R but also a small part of region surrounded R in the input feature map. That is, input data regions of the adjoining partitions will overlap with each other in the feature space [6]. The outward offset for the partition is determined by the size of convolution kernel, and can be given as:

$$W_{offset} = \frac{K-1}{2} \tag{3}$$

Depending on the total number of fused layers n ($n \geq 1$) in a block, the actual range $[Index^i_{start}, Index^i_{end}]$ for the partition i can be given as,

$$Index^i_{start} = \begin{cases} 0 & i = 1 \\ \sum_{j=1}^{i-1} W_j - n * W_{offset} & i \neq 1 \end{cases} \tag{4}$$

$$Index^i_{end} = \begin{cases} \sum_{j=1}^{i} W_j + n * W_{offset} & i \neq M \\ W & i = M \end{cases} \tag{5}$$

Note that FN 1 and FN M located at the border have no neighboring partition on one side. We solve this problem in the computation process by padding zeros at the vacant side as overlapped elements. Although the overlap between input partitions leads to additional costs on communication and computation, we believe that these costs are affordable and much less than those in the layer-wise strategy.

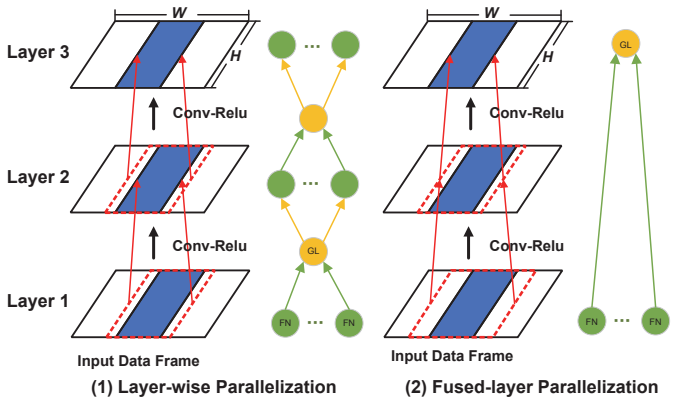

Fig. 3. Layer-wise and fused-layer partitioning strategies.

IV. Performance Evaluation

A. Experiment Settings

The implementation of EdgeLD is based on PyTorch, which is a Python based tensor library for deep learning research. As suggested in [17], [18], we create multiple virtual machines (VMs) with constrained resources to emulate our edge devices. All VMs have a single-core 800MHz CPU, 1GB of memory, and run Ubuntu 18.04. The networking module for inter-VM communication is implemented using Python Socket APIs, and the Linux Traffic Control (TC) command is used to configure the inbound/outbound network bandwidth for all VMs. Without loss of generality, we evaluate EdgeLD using the VGG13 and VGG16 models, which are two variations of the popular VGG network architecture [3], [7].

We first run profiling experiments to the VMs, with key parameters $H, W \in \{7, 14, 21, \ldots, 217, 224\}$, $K = 3$, and $C_{in}, C_{out} \in \{32, 64, 128, 256, 512\}$. The results shown in Fig. 4 verify the linear relationship between computation time and FLOPs. The regression model is finally obtained as $T^{conv}_{comp} = 6.24E{-}11 * F + 1.97E{-}2$.

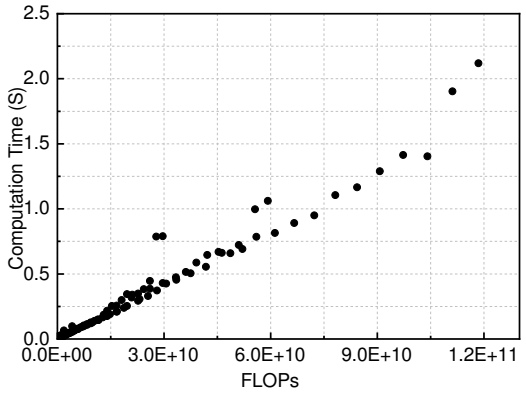

Fig. 4. FLOPs versus computation time.

Without otherwise specified, in the following experiments, the number of VMs is 5 (i.e., 1 GL and 4 FNs, the same with [3], [7]), the size of input feature map is $224 * 224$, the

TABLE I
EXECUTION TIME COMPARISON BETWEEN BODP AND OCBP (UNIT: SECOND).

Network Models		VGG13				VGG16			
Network Bandwidth	FNs	1	2	3	4	1	2	3	4
100Mbps	BODP	22.29	12.84	8.07	7.64	24.57	14.07	9.50	8.78
	OCBP	20.72	12.04	7.34	6.84	24.02	13.71	9.32	7.52
	Gain	7.04%	6.23%	9.04%	8.31%	2.24%	2.56%	1.89%	14.34%
1000Mbps	BODP	21.281	12.798	8.28	7.30	23.69	13.5	9.22	7.58
	OCBP	20.71	11.25	6.97	6.26	23.36	13.23	8.37	6.53
	Gain	2.58%	12.10%	15.82%	14.25%	1.39%	2.00%	9.22%	13.58%

number of input data frame is 1, and the inter-VM bandwidth is 1000Mbps. The OCBP scheme is applied to partition the input computation workload, and the layer fusion technique is used for efficient data exchanging between FNs and GL.

B. Evaluation of Model Partitioning Scheme

In this set of experiments, we compare the model partition scheme OCBP in EdgeLD against the BODP in MoDNN [3]. Table I illustrates the comparison results on model execution time with typical Ethernet bandwidth and different number of FNs. For VGG13/VGG16 models, our OCBP outperforms BODP in terms of the execution time by 2.68%-15.82% and 1.39%-14.34%, respectively. That's because BODP does not take into consideration the impact of network bandwidth in workload assignment, resulting in imbalanced partitioning and idle waiting among FNs, especially when the number of FNs is relatively large.

From Table I, we can infer that high bandwidth can help to reduce the time consumption. Therefore, we further run more experiments under various bandwidth settings. As depicted in Fig. 5, OCBP has an execution speedup of 2.68-3.77×, always better than BODP (2.56-3.19×), especially when the network is fast. The results confirm the advantages of taking bandwidth into account, and validate the efficiency of the OCBP scheme used in EdgeLD.

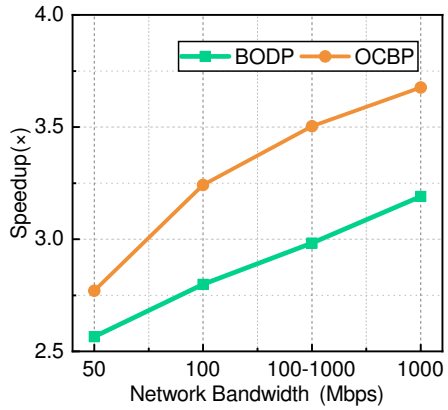

Fig. 5. Speedup comparison between BODP and OCBP schemes.

C. Evaluation of Data Exchanging Strategy

Fig. 6 shows how the communication cost is reduced by layer fusion. In the experiments, multiple consecutive convolutional layers between two pooling layers are treated as a fused block. Thanks to the contribution of layer fusion, compared to the layer-wise strategy, the total execution achieves a 1.11× and 1.13× speed boost for VGG13 and VGG16, respectively. We believe that the performance improvement will be more remarkable when EdgeLD is applied to deeper neural network structures [2].

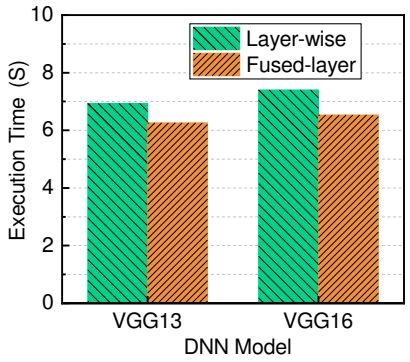

Fig. 6. Execution time cost comparison between layer-wise and fused-layer strategies.

D. Impact of System Parameter Variation

1) Impact of the size of feature map: Fig. 7 and Fig. 8 show the impact of the size of input feature maps on inference speed of VGG13 and VGG16, respectively. Following the increase of input data size, the total execution time grows explosively from about one second to tens or even hundreds of seconds. Generally, the growth rate of time consumption is much slower when more FNs are used for parallel execution. Taking VGG13 as an example, it takes about 14 times and 214 times longer, respectively, for a FN to process feature maps of size 224*224 and 448*448 than that of 96*96. When using 4 FNs, we can find that the corresponding time expenditure only increases by 10 times and 81 times, respectively. However, we can observe a sharp increase in time and a sudden decrease in speedup from Fig. 8, when the feature map size is 448*448. The reason is that the memory usage of VGG16 has exceeded the allocated capacity of FN memory, and the Linux swap partition is used to help the system handle extra load. Note that the access time for swap is much slower than that of physical memory, so the CPU has to spend a lot of time idle waiting for data to be ready. This observation indicates that the memory capacity of FNs has an important impact on computation efficiency, especially when the DNN model is relatively large and complex.

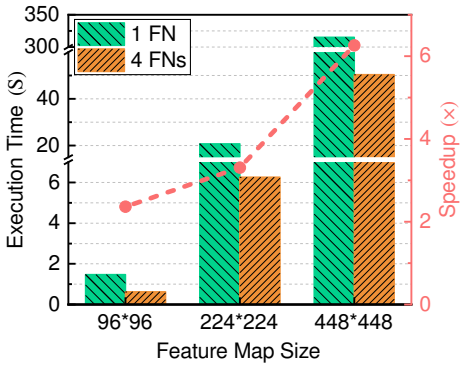

Fig. 7. Performance comparison of VGG13 with different size of input feature map.

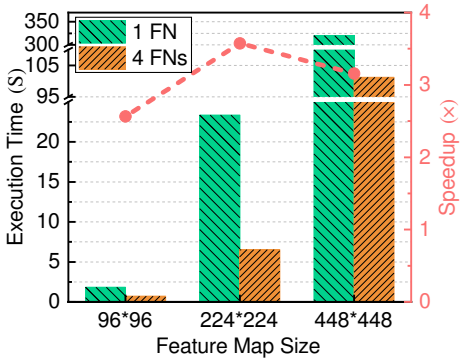

Fig. 8. Performance comparison of VGG16 with different size of input feature map.

2) Impact of the number of data frames: Fig. 9 and Fig. 10 show the impact of the number of input data frames on inference speed of VGG13 and VGG16, respectively. When the number of input data frames increases from 1 to 5, the total execution time grows steadily for both VGG13 and VGG16. Meanwhile, we can notice that the speedup ratio of VGG13 first increases from 3.3 to 5.4, and then decreases to 3.1. Under the same settings, the speedup ratio of VGG16 follows a decreasing trend from 3.57 to 2.62. All these results reveal that it is not a good choice to input too much data at one time. The reason is that the physical memory capacity is limited for edge devices, and the computation will be forced to slow down due to the I/O cost of swapping partitions between memory and disk.

3) Impact of the number of FNs: Fig. 11 shows the speedup achieved by EdgeLD with different number of FNs. From this figure, the inference speed will increase monotonically linearly with the number of FNs. For VGG13, the total execution time in EdgeLD reaches 1.84-3.30× speedup with the number of FNs increasing from 2 to 4. For VGG16, the total execution time in EdgeLD improves by 1.77×, 2.79×, and 3.57× with 2, 3, and 4 FNs, which are much higher than those (1.53×, 1.67×, and 2.03×) reported in MoDNN [3].

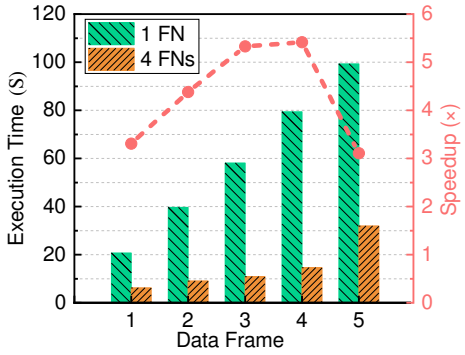

Fig. 9. Performance comparison of VGG13 with different number of data frames.

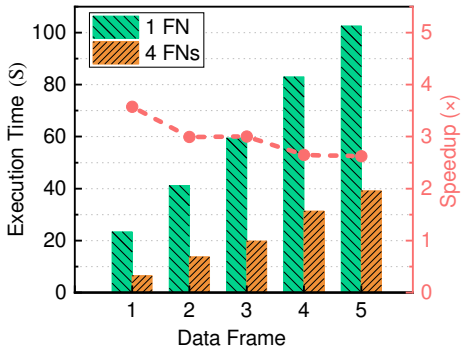

Fig. 10. Performance comparison of VGG16 with different number of data frames.

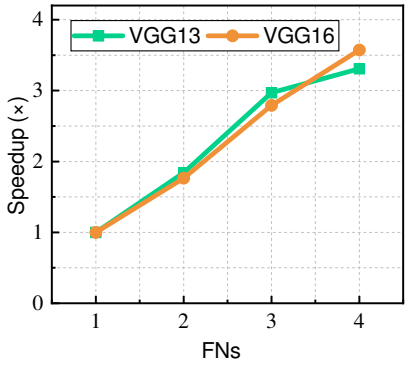

Fig. 11. Speedup with different number of FNs.

V. Conclusion

In this paper, we propose a local distributed DNN inference framework, EdgeLD, to enable computation parallelism with resource-limited edge devices. EdgeLD can dynamically and flexibly partition a DNN model for parallel execution to adapt to heterogeneous computing resources and different network conditions. Experimental results show that EdgeLD achieves nearly linear performance improvement for DNN inference, and outperforms existing schemes under various parameter settings. Future work will include optimizing the execution of other network layers, improving fault-tolerance of model

partitions, and generalizing the framework for more types of DNN models and edge devices. The hope is that frameworks like EdgeLD can enable edge intelligence and promote AI-based IoT (AIoT) applications in the near future.

REFERENCES

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug 2019.

[2] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[3] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 1396–1401.

[4] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 1135–1143.

[5] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1423–1431.

[6] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[7] J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. W. Nixon, X. Chen, H. Li, and Y. Chen, "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)*, 2017, pp. 751–756.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016*, 2016.

[9] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2654–2662.

[10] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[11] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1800–1807.

[12] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 17. New York, NY, USA: Association for Computing Machinery, 2017, p. 615629.

[13] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, 2018, pp. 671–678.

[14] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 122–138.

[15] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *arXiv preprint arXiv:1611.06440*, 2016.

[16] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson, "Effects of communication latency, overhead, and bandwidth in a cluster architecture," *SIGARCH Comput. Archit. News*, vol. 25, no. 2, p. 8597, May 1997.

[17] M. Javaid, A. Haleem, R. Vaishya, S. Bahl, R. Suman, and A. Vaish, "Industry 4.0 technologies and their applications in fighting covid-19 pandemic," *Diabetes & Metabolic Syndrome: Clinical Research & Reviews*, vol. 14, no. 4, pp. 419 – 422, 2020.

[18] L. Zhou, H. Wen, R. Teodorescu, and D. H. Du, "Distributing deep neural networks with containerized partitions at the edge," in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*. Renton, WA: USENIX Association, Jul. 2019.