Accelerating Convolutional Neural Network Inference in Split Computing: An In-Network Computing Approach

Hochan Lee
Korea University
Seoul, Korea
ghcks1000@korea.ac.kr

Haneul Ko
Kyung Hee University
Yongin, Korea
heko@khu.ac.kr

Chanbin Bae
Korea University
Seoul, Korea
bin6050@korea.ac.kr

Sangheon Pack
Korea University
Seoul, Korea
shpack@korea.ac.kr

***Abstract*—Since the latest deep neural network (DNN) models are complex and have many layers, processing an entire DNN model on mobile devices is challenging. To cope with this challenge, a split computing (SC) approach has been proposed, which divides a DNN model into multiple layers and distributes them to mobile devices and edge servers. On the other hand, in-network computing (INC) is a promising technology that offloads computational tasks to network devices (e.g., programmable switches) and thus provides low latency and line-rate packet processing. Although the switch cannot directly process complex DNN models due to its limited computing and memory resources, it has the potential to process specific layers that require simple arithmetic operations. For example, processing the max-pooling layer of convolutional neural network (CNN) models can be offloaded to the switch. In this paper, we consider a network where there are three types of computing nodes: mobile device, edge servers, and switches, and formulate the problem of placing the layers of the CNN model on the computing nodes to minimize the inference latency considering the resource constraints of computing nodes. Then, we derive the optimal results by solving the formulated optimization problem. Evaluation results demonstrate that the optimal results show lower inference latency than a random layer placement scheme and a server-only placement scheme.**

***Index Terms*—Split computing, In-network computing**

I. Introduction

Recent advancements in deep learning (DL) applications, such as computer vision [1], [2] and natural language processing [3], have shown outstanding performance improvements. These improvements have been achievable due to the development of high-performance hardwares (e.g., GPUs and TPUs), which enables the training of larger and deeper deep neural network (DNN) models. However, deploying these models on mobile devices has become challenging due to the limited computing and memory resources of mobile devices.

To address this challenge, split computing (SC) [5] has gained significant attention. It divides large DNN models into multiple segments (e.g., head and tail models) and distributes processing tasks across mobile devices and edge servers [4]. In this approach, mobile devices are responsible for processing only a portion of the model for forward propagation, transmitting intermediate results to the edge servers. The final inference results are generated at the edge server by taking the intermediate results as input to the rest of the model. SC significantly reduces communication overhead, since mobile devices only need to transmit intermediate data rather than all input data. Furthermore, the processing at the edge servers offers lower latency performance compared to traditional methods based on cloud computing.

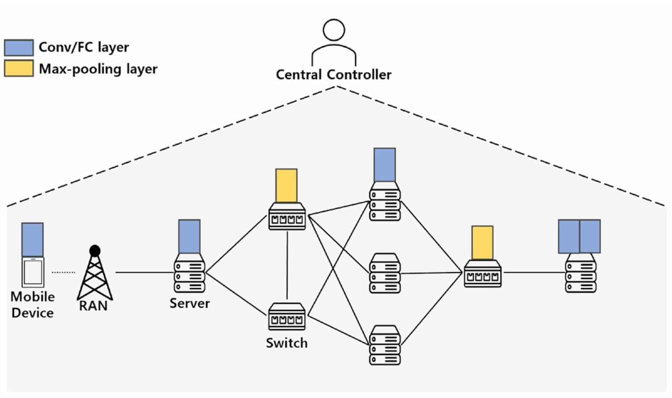

Fig. 1: System model.

In-network computing (INC) technology has also attracted attention in recent years. INC is a technology that offloads computational tasks that were traditionally performed at end hosts to network devices, such as programmable switches [6]. Using INC, it is possible to decrease latency by executing computations on the programmable switches in the forwarding path rather than on remote servers. In addition, INC takes advantage of the high-performance processing speed of programmable switches, enabling packet processing at line-rate speeds.

Unfortunately, because of the limited computing and memory resources of programmable switches, it is still difficult to directly process a large DNN model on the switch. However, it is possible to offload the processing of some specific layers of the DNN model to the switch. For example, convolutional neural network (CNN) models, which are widely used in the image processing field, have several max-pooling layers that are used for feature extraction and spatial reduction.

Since these max-pooling layers require only simple arithmetic operations such as the max function, programmable switches can process the max-pooling layer. Because the processing speed of switches is hundreds to thousands of times faster than that of servers, using INC in SC can lead to a significant reduction in the inference latency.

In the SC environment where mobile devices, switches, and edge servers are mixed as computing nodes, how to place CNN layers on computing nodes considering resource constraints should be addressed to minimize the CNN inference latency. Although programmable switches have high processing speeds, they have limited memory and thus can handle only one max-pooling layer. Therefore, it is crucial to consider such different memory and computing constraints of computing nodes during a CNN layer placement.

In this paper, we consider a network where three types of computing nodes (i.e., a mobile device, edge servers, and switches) can process parts of CNN layers, and then formulate the CNN layer placement problem which aims to minimize the CNN inference latency experienced by the mobile device considering resource constraints of computing nodes. After that, we derive the optimal results by solving the formulated optimization problem. From the evaluation results, it can be demonstrated that the optimal results show lower inference latency than a random layer placement scheme and a server-only placement scheme.

II. System Model

Figure 1 shows the system model of this paper, which consists of a central controller and computing nodes. Computing nodes include a mobile device, edge servers, and switches. In the system model, the mobile device sends a request to the controller for CNN inference to utilize the inference results in their applications. To perform CNN inference requested by the mobile device on the network, the controller adopts the SC approach. The controller splits a CNN model into multiple layers and places each layer on several computing nodes to minimize the inference latency.

A network topology is defined as an undirected graph $G = (\mathbf{N}, \mathbf{E})$, where $\mathbf{N}$ is a set of computing nodes denoted by $\mathbf{N} = \{n_1, n_2, ..., n_N\}$ and $\mathbf{E}$ is a set of links between nodes denoted by $\mathbf{E} = \{e_1, e_2, ..., e_E\}$. We consider three types of computing nodes: a mobile device, an edge server, and a switch. The mobile device and edge servers can process all CNN layers, while switches can process only one of the max-pooling layers.

Each computing node $n \in \mathbf{N}$ has two types of resource capacity: computing and memory capacity. The computing capacity and the memory capacity of the computing node n are described as c_n^{com} and c_n^{mem}, respectively. On the other hand, each link $e \in \mathbf{E}$ has a fixed amount of bandwidth b_e. We assume that the controller knows a global network view and can calculate the link bandwidth between any nodes in the network.

The set of CNN layers is denoted by $\mathbf{L} = \{l_1, l_2, ..., l_L\}$. Layer $l \in \mathbf{L}$ can be a convolutional layer, a max-pooling layer, or a fully-connected layer. A binary variable M_l indicates whether the layer l is a max-pooling layer or not. Each layer has different requirements for memory and computing capacity. The memory capacity required to store the layer l is denoted by r_l^{mem} and the computing capacity required to process the layer l is denoted by r_l^{com}. In addition, they yield different sizes of output depending on the layer architecture. The size of the output results produced by the layer l is denoted by o_l. A binary variable $X_{n,l}$ indicates whether the layer l is placed on the computing node n or not.

TABLE I: Summary of notations.

Notation	Description
$\mathbf{N}$	Set of computing nodes
$\mathbf{L}$	Set of DNN layers
$\mathbf{E}$	Set of links
c_n^{com}	The computing capacity of the computing node n
c_n^{mem}	The memory capacity of the computing node n
r_l^{mem}	The memory capacity required to store the CNN layer l
r_l^{com}	The computing capacity required to process the CNN layer l
o_l	The output data size of the CNN layer l
b_e	The bandwidth of the link e
$b_{n,n'}$	The minimum link bandwidth between computing nodes n and n'
$h_{n,n'}$	The minimum number of hops between the computing nodes n and n'
d	The CNN inference latency
d_l	The total propagation latency
d_p	The total processing latency
$X_{n,l}$	Equal to 1, if the computing node n has the CNN layer l
M_l	Equal to 1, if the CNN layer l is the max-pooling layer
S_n	Equal to 1, if the computing node n is the switch

III. Problem Formulation

In this section, we formulate the CNN layer placement problem to minimize the inference latency in the system model. To this end, we first present the objective function of the problem. After that, we explain several constraints that should be considered in the problem. We summarize the notations used in this paper in Table I.

A. Objective Function

Our goal is to minimize CNN inference latency experienced by a mobile device. The CNN inference latency is denoted by d and calculated by

$$d = d_t + d_p, \tag{1}$$

where d_t and d_p refer to the total transmission latency and the total processing latency, respectively.

The total transmission latency d_t is defined as the sum of the time it takes for all computing nodes to transmit the output data to the next computing node and is calculated by

$$d_t = \sum_{n \in N} \sum_{n' \in N} \sum_{l \in L} \frac{o_l h_{n,n'} X_{n,l} X_{n',l+1}}{b_{n,n'}}, \tag{2}$$

where $b_{n,n'}$ is the minimum bandwidth among links between the computing nodes n and n', and $h_{n,n'}$ is the minimum number of hops between the computing nodes n and n'.

The total processing latency d_p is defined as the sum of time it takes for all nodes to process the assigned CNN layers and is calculated by

$$d_p = \sum_{n \in N} \sum_{l \in L} \frac{r_l^{com} X_{n,l}}{c_n^{com}}, \quad (3)$$

where c_n^{com} is the computing capacity of the computing node n. Therefore, the objective function can be expressed as

$$\min_{X_{n,l}} d \quad (4)$$

$$\text{s.t.}(6) - (9). \quad (5)$$

B. Constraints

The layer l should be placed on only one computing node. Therefore, we have

$$\sum_{n \in N} X_{n,l} = 1. \quad (6)$$

For the switches, they cannot process convolutional or fully connected layers, but can only process one max-pooling layer. On the other hand, the mobile device and the edge servers can process any type of layer and can accommodate multiple layers. This constraint can be described as

$$\sum_{l \in L} S_n M_l X_{n,l} + (1 - S_n) \leq 1. \quad (7)$$

Since each computing node has a fixed memory capacity, the sum of the memory capacities required to store the CNN layers deployed on the computing node n should not exceed c_n^{mem}. Thus, we have the following constraint as

$$\sum_{l \in L} r_l^{mem} X_{n,l} \leq c_n^{mem}. \quad (8)$$

Similarly, each computing node has a fixed computing capacity, and the maximum value of processing capacities required to process the CNN layers deployed on the computing node n should not be greater than c_n^{com}. This constraint can be expressed as

$$\max_{l \in L} (r_l^{com} X_{n,l}) \leq c_n^{com}. \quad (9)$$

IV. Simulation Results

For performance evaluation, we configure a simple network topology consisting of a mobile device, two edge servers, and two switches as in Figure 2. Due to the large complexity of calculating an optimal solution to the formulated problem, we consider a simplified CNN model based on the layer specification in AlexNet [9][1]. The CNN model considered has 5 layers, and its specification is shown in Table II, and the input data size is set to 5MB.

The default memory and computing capacity of the edge servers are set to 700MB and 700MFLOPS, respectively. Those of the mobile device are set to be values 10 times smaller than the edge server, respectively. Lastly, the memory capacity of switches is set to the same value as the memory capacity required to process a max-pooling layer. To configure the computing capacity of switches, we measured the processing latency of processing the 2x2 max-pooling layer in a hardware programmable switch (i.e., Intel Tofino [8]). The measured latency is in the hundreds of nanoseconds, which is much smaller than the processing latency of edge servers, which is typically in the tens to hundreds of milliseconds. Therefore, we set the processing latency of the switches to be negligible by setting the computing capacity of the switches to a much higher value than that of the edge server. The default link bandwidths between computing nodes are set to 5Gbps, except for the wireless link, which is set to 500Mbps.

[1]In future work, we will design an algorithm that can run in a polynomial time to address a complex CNN model and network topology.

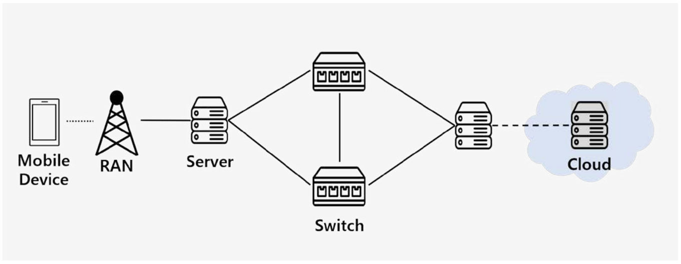

Fig. 2: Simulation topology.

TABLE II: Specification of the CNN model in the evaluation.

Layer Index l	1	2	3	4	5
Type	Conv	Max	Conv	Max	FC
r_l^{com}(FLOPS)	100M	50M	200M	50M	300M
r_l^{mem}	1GB	0.25GB	5GB	0.25GB	0.5GB
o_l	10MB	2.5MB	6MB	1.5MB	1MB

We calculate optimal results by solving the CNN layer placement problem formulated with a brute-force algorithm (called OPTIMAL), and we compare the optimal results with the following schemes: 1) RANDOM adopts random placement of CNN layers; 2) SERVER finds the best placement strategy by using only edge servers as computing nodes without the usage of switches; 3) CLOUD places entire CNN model on a remote cloud, which has unlimited memory capacity. We assumed that the remote cloud has 5GFLOPS of computing capacity and the link bandwidth between the mobile device and the cloud is set to 200Mbps.

A. Effect of the computing capacity of edge servers

Figure 3 shows the inference latency of all schemes depending on the computing capacity of the edge servers. It can be found that OPTIMAL shows the lowest inference latency performance, except for the case where the computing capacity is 300MFLOPS. This is because it is beneficial to process the entire CNN model in the cloud with significant computing capacity if the computing nodes have too little computing capacity. On the other hand, SERVER uses only edge servers to process CNN layers, and thus does not benefit from high-speed processing switches, allowing switches to only transmit intermediate results. Meanwhile, since RANDOM does not

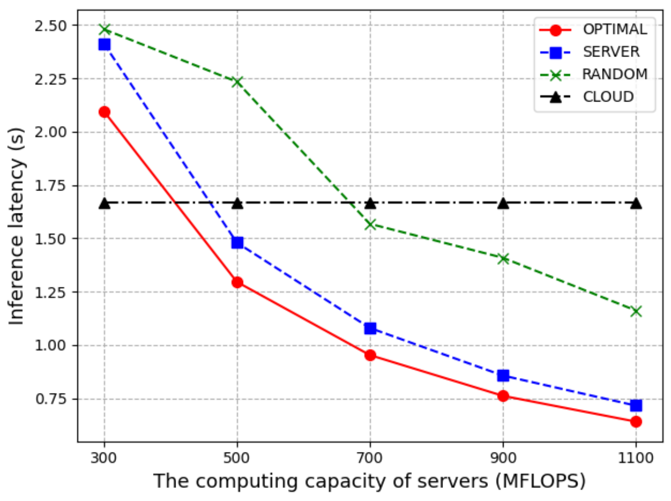

Fig. 3: The effect of the computing capacity of edge servers.

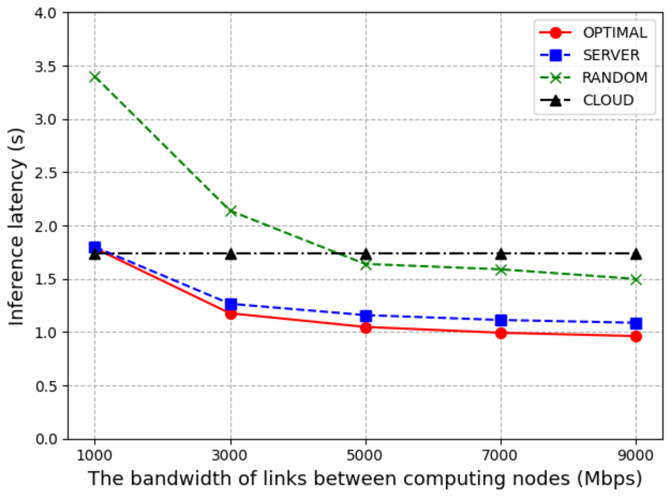

Fig. 4: The effect of the link bandwidth between computing nodes.

consider the order of layer processing, it can cause unnecessary transmission latency to process layers sequentially, resulting in a larger inference latency than SERVER.

B. Effect of the link bandwidth between computing nodes

Figure 4 shows the effect of the link bandwidth between computing nodes. It is observed that as the bandwidth increases, there is a slight increase in the performance gap between OPTIMAL and SERVER. This is because the transmission latency required to reach the switches is decreased, and thus, the benefit of low processing latency by the switches increases. Similarly, it can be found that the performance gap between RANDOM and SERVER decreases as bandwidth increases, this is because RANDOM can also utilize switches as computing nodes, unlike SERVER.

V. Conclusion

In this paper, we formulated a CNN layer placement problem to minimize inference latency in an SC environment where there are three types of computing nodes: mobile device, servers, and switches. Then, we solved the formulated problem and derived the optimal results. Lastly, we compared its latency performance with other heuristic schemes. However, since the optimal solution has a large computation complexity, a novel heuristic algorithm that shows near-optimal performance but low computation complexity must be deployed in practical environments. In our future work, we will design a new algorithm that can run in a polynomial time considering a topological relationship between computing nodes and will evaluate the proposed algorithm in more practical environments, including hardware edge servers and programmable switches.

Acknowledgments

This work was supported by Institute for Information and communications Technology Planning & Evaluation (IITP) grant (IITP 2022-0-00531) and in part by National Research Foundation (NRF) (No. 2020R1A2C3006786) funded by the Korean government.

References

[1] A. Krizhevsky, I. Sutskever, and GE. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS 2012*, December 2012.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. IEEE CVPR 2016*, June 2016.
[3] J. Devlin, M. Chang, K, Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL 2019*, June 2019.
[4] J. Chen, Q. Qi, J. Wang, H. Sun, and J. Liao, "Accelerating DNN Inference by Edge-Cloud Collaboration," in *Proc. IEEE IPCCC 2021*, October, 2021.
[5] Y. Matsubara, M. Levorato, and F. Restuccia, "Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1-30, December 2022.
[6] R. Bifulco and G. Retvari, "A Survey on the Programmable Data Plane: Abstractions, Architectures, and Open Problems," in *Proc. IEEE HPSR 2018*, June 2018.
[7] P4 Language Consortium. [Online]. Available: https://p4.org/specs/
[8] Intel Tofino Series Programmable Ethernet Switch ASIC. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html
[9] 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 *Proc. ASPLOS 2017*, April 2017.