DeePar: A Hybrid Device-Edge-Cloud Execution Framework for Mobile Deep Learning Applications

Yutao Huang[1], Feng Wang[2], Fangxin Wang[1], Jiangchuan Liu[1]
[1]School of Computing Science, Simon Fraser University, Canada
[2]Department of Computer and Information Science, The University of Mississippi, USA

***Abstract*—With the deep penetration of mobile devices, more and more mobile deep learning applications have been widely used in daily life. However, since deep learning tasks are computationally intensive, the limited computation resource on mobile devices cannot execute the application effectively. The common approaches are transmitting the data from mobile devices and offloading the computation to the cloud. This brings another issue that the high data transmission delay may become the bottleneck of the performance. In this paper, we explore a new rising concept, edge computing, into mobile deep learning applications. Comparing with cloud computing, the communication delay can be significantly reduced. To this end, we note that there exists a layer-level partitioning strategy for deep neural networks to distribute the computation loads more smoothly among the device, the edge and the cloud, which can further reduce the overall execution delay. We propose a framework called DeePar which exploits all the available resources from the device, the edge server, and the cloud to collaboratively optimize the inference performance. We also formulate a scheduling problem for the multi-task execution and propose an efficient solution. Both our prototype experiments and extensive simulations show that DeePar can achieve up to 80% delay reduction.**

I. Introduction

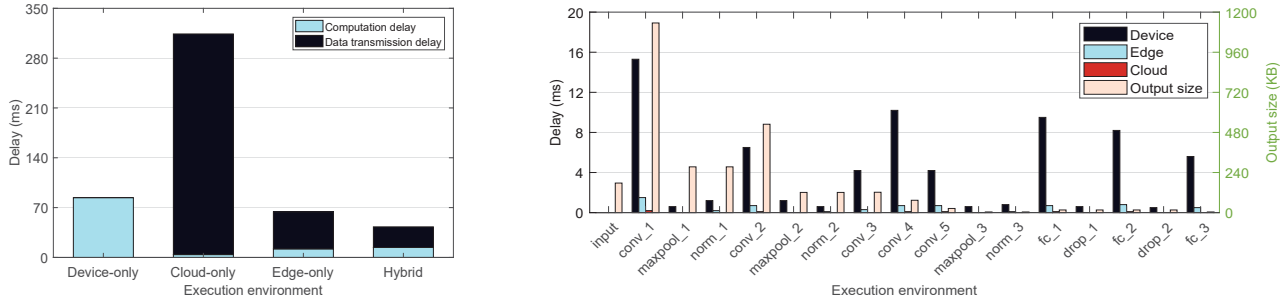
In recent years, machine learning, particularly deep learning has attracted significant attentions from both industry and academia, where the key component of deep learning, the Deep Neural Networks (DNNs) have significantly innovated the state-of-the-art techniques in computer vision [1], pattern recognition [2] as well as other research fields. On the other hand, the deep penetration of mobile devices and applications in our daily life have called upon more and more demands for them to interact with deep learning techniques. Some popular mobile applications, including Apple Siri, FaceID and Google Now, are by default integrated with mobile systems, where the key modules are implemented with deep learning models.

Though shown great potentials, mobile deep learning applications are facing the challenges with the low capacities of mobile devices and high resource demands of deep learning tasks. One commonly used approach to solve this problem is to reroute the input data from mobile devices and offload the intensive deep neural network processing remotely to the cloud. However, despite that major cloud providers like Amazon, Google and Microsoft all start to provide convenient cloud services for deep learning applications, the concerns of the high communication cost and the high data transmission latency over the Internet as well as the privacy issue therein are still hindering the quality of users' experiences.

On the other hand, the emerging concept of edge computing brings new opportunities to offer delay-sensitive and cost-efficient services. The concept of edge is referred to the network edge, which is in close proximity to the end devices. Comparing to the cloud, the communication latency is much lower between the edge and the mobile device. By offloading the data, applications and services to the near-end edge servers, the network transmission delay can be greatly reduced [3]. Although the edge computing has plenty of advantages over the cloud computing, performance issues may still exist if the deep learning applications are solely supported by edge computing, especially when a large number of application requests are involved. Since the computation and storage resources of edge servers are often limited and less powerful comparing to cloud servers, making the deep learning network design have to be comprised to trade off between the achievable performance and the available resource. Therefore, having an all-round scalable, responsive, and cost-effective solution for mobile deep learning applications still remains an illusive goal.

In this paper, we take a close investigation of several typical state-of-the-art deep learning models, revealing that instead of executing the entire deep learning inference process either on the device, the edge server or the cloud, the execution load can actually be carefully split among all of them with a layer-level partitioning strategy based on the hierarchical structure of DNNs. To this end, we propose a framework named DeePar which can hybridly exploit all the available resources from the device, the edge server and the cloud to unleash the full power of the deep learning networks therein. To maximize the achieved performance while still keeping it cost-effective, we further formulate a multi-task partitioning optimization problem for our DeePar framework as well as propose a smart and efficient scheduling solution. We evaluate our DeePar solution with both prototype experiments and extensive simulations. The results demonstrate our DeePar can achieve up to 80% inference delay reduction.

The remainder of this paper is organized as follows. In Section II, we discuss the background and motivation to partition deep learning networks in mobile applications to hybridly run on the device, the edge server, and the cloud. Section III provides an overview of our DeePar framework and formulates the multi-task partitioning problem to maximize the achieved performance while still being cost-effective. We then propose a smart scheduling solution to efficiently address the formulated problem in Section IV. Section V describes the



(a) AlexNet Performance Comparison

(b) AlexNet Layer-level Performance

Fig. 1: AlexNet Delay Performance

experiment and simulation setup and presents the evaluation results. Finally, we conclude the paper in Section VI.

II. Background and Overview

In this section, we discuss the background of deep learning over the network and the performance of DNNs in a layer-level perspective which motivates our work.

A. Mobile Deep Learning with the Cloud and the Edge

Mobile deep learning applications are facing the challenges of limited and less powerful computation resource on mobile devices, where the developers commonly choose to push the computation process on the remote cloud [4]. However, due to the high latency and the limited bandwidth of the network connection between the device and the cloud, the data transmission delay has become a bottleneck especially for applications involving with large-scale datasets. Recently, edge computing has been proposed to provide a new opportunity to overcome this challenge. The concept of edge is referred to the network topology in close proximity to the end device which is near the wireless access points such as Wi-Fi routers and 4G/5G base stations. By deploying servers which are equipped with computation and storage resource at the edge, deep learning models and input data can be pushed to the edge server, resulting in a significant reduction in the entire inference delay compared to conventional cloud-based approaches [5].

There are also continuing research efforts to identify appropriate offloading strategies based on deep learning application's own characteristics. For example, Kang *et al.* proposed Neurosurgeon [4], which considered mobile devices as edge hosts and only investigated the partitioning of one mobile deep learning application task between the mobile devices and the cloud. Different from these aforementioned efforts, we seek a more general and flexible framework that can well balance and optimize the overall performance of all the considered mobile deep learning application tasks across mobile devices, edge servers and the cloud.

B. DNN Execution with Layer-level Partitioning Optimization

To fully utilize the convenient edge and the high-performance cloud to further improve the overall DNN inference performance, we propose to a hybrid offloading approach to collaboratively utilize all the resources available on the device, the edge, and the cloud. Instead of partitioning the neural network into two parts, we bring the power of edge servers into our framework. By introducing one extra partition of the neural network, we investigate the possibility and opportunity to run separate parts of the inference process on the device, the edge server and the cloud collaboratively, which can better improve the overall performance.

To validate the feasibility of our layer-level partitioning strategy, we further conduct a real-world case study to verify the effectiveness of our idea. To this end, we choose the AlexNet [6] as our DNN model, which is a representative state-of-the-art Convolutional Neural Network for image classification. The experiment setup environment is stated in Section V. We first execute the entire DNN inference separately on the device, the edge, and the cloud. The total delays are shown in Fig. 1(a). We observe that the execution on the cloud has the longest average delay (314 ms) per sample due to the extremely high data communication delay, while the execution on the edge has a smaller delay (64 ms) per sample compared to the execution on the mobile device which takes 84 ms.

Then we investigate in the layer-level execution performance of AlexNet, which is measured on our edge server. The results are shown in Fig. 1(b), where the x-axis describes each layer in the AlexNet, while the average computation delay and the output data size for each layer are shown as bar charts respectively. We can see that the major computation delays are contributed by Convolutional Layers (shown as *conv_x*) and Fully-Connected Layers (shown as *fc_x*), while the computation delays for Max-Pooling Layers, Normalization Layers and Dropout Layers (shown as *maxpool_x*, *norm_x* and *drop_x* respectively) are negligible when comparing with Convolutional Layers and Fully-Connected Layers. From the perspective of the output size, we can find that Convolutional Layers will usually increase the output size and Max-Pooling Layer will significantly reduce the output size.

Based on the above observations, we can infer that making partitions either before the Convolutional Layer or the Fully-Connected Layer, as well as after the Max-Pooling Layer, may be effective to reduce computation and data communication delay. To this end, we further conduct experiments for each feasible partitioning scheme and find out that making two partitions after layer *maxpool_1* and *maxpool_3*, and hybridly executing the resulted three parts on the device, the edge server and the cloud respectively will yield the shortest overall

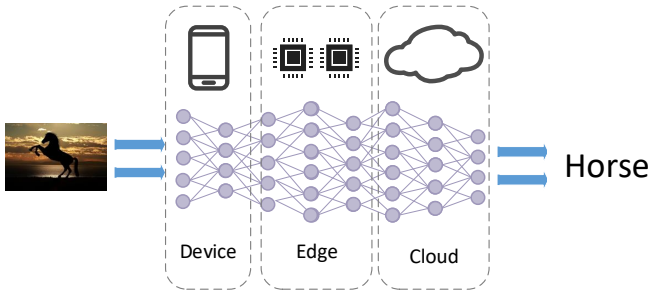


Fig. 2: DeePar Framework

inference delay (43ms) as shown in Fig. 1(a).

C. DeePar: a Hybrid Execution Framework

Motivated by our observations and experiment results, we propose DeePar, a hybrid device-edge-cloud execution framework to fully unleash the power of the deep leaning in mobile applications. As illustrated in Fig. 2, we make two partitions which push the front, the intermediate and the last remaining part of the model inference to the mobile device, the edge server, and the cloud, respectively. The input of the partial DNN model on the edge server (cloud) is exactly the same as the output generated on the device (edge server). After finishing three stages of the computation, the final inference result will be generated as the last layer's output and transmitted back to the device.

Since the processing power increases when moving the computation from the device to the edge and from the edge to the cloud, the performance gain of the total inference delay can be obtained if the reduction of computation delay on the higher-performance computation platform is greater than the transmission delay of the intermediate layer's output. To better understand the partitioning gain and design appropriate algorithms to maximize the advantage of our DeePar framework, we further formulate it as a mathematical optimization problem in the next section.

III. Problem Statement

In this section, we formulate an optimization problem based on our DeePar framework to further maximize its benefits. Since our target is on mobile deep learning applications, we consider each DNN inference task is generated by a mobile device. Currently, there are multiple communication standards that can be selected as the mobile network, for example, Bluetooth, Wi-Fi, 3G/4G/LTE or even 5G network. If edge servers are deployed at the gateway or base station for each accessible connection, then multiple choices are available for the task execution. Therefore, our formulated problem should not only determine the optimal DNN partitioning, but also consider the appropriate scheduling of the partitioned deep learning tasks across the device, the edge server and the cloud.

Suppose there are n mobile devices, m edge servers and one centralized cloud platform. Each mobile device can connect to one of the connectable edge servers, and each edge server is connecting to the cloud through the backbone network. Each edge server and the cloud is implemented on virtual machines(VMs) or containers in the physical servers equipped with all types of resources including GPU, CPU, memory and disk storage. The computation resource on each edge server, as well as the bandwidth resource on each edge server and the cloud is limited. For simplicity, we assume that a mobile device only generates one deep learning task i, where the total number of data samples involved is denoted as N_i.

For the network resources, each job will occupy reserved uplink bandwidth for either the connections between the mobile device and the edge server or the connections between the connected edge server and the cloud, to guarantee data transfer performance. Once the connection is established, the reserved bandwidth will not be allowed to change. Bandwidth reservation for a VM or container is common for accelerated computing in cloud platforms [7], e.g., the reserved bandwidth of EC2 GPU instance P2 on AWS is 10 Gbps or 20 Gbps. For each job i, we define the bandwidth of the two links as b_i^1 and b_i^2. Also, due to the limitation of hardware resources and the specific requirements of each task, the bandwidth will have an upper bound for allocation and we defined it as B_i^1 and B_i^2. The allocable maximum downlink or uplink bandwidth for each edge server is denoted as B_e^1 and B_e^2 respectively. Since the edge server locates near the network gateway, the link connecting each mobile device to the cloud should include one edge as the intermediate hop.

For the computation and storage resources, we assume the cloud provides abundant computation resources which can support all tasks' execution. Different from the cloud, each edge server has finite computation and storage resource to allocate. To this end, we use r_e to denote the maximum number of concurrent tasks that can be executed on each edge server, $\forall e \in E$.

For the edge network connection, x_{ie} and y_{ie} are defined as the edge network occupation and core network occupation indicator respectively. Also, z_{ie} is defined as the edge resource utilization indicator. Each mobile device $\forall i \in I$ can only connect to a subset of all the edge servers. We define this subset as $E_i \subseteq E$.

Because of the hierarchical structure of a neural network, each layer's computations only can be performed after completing the previous layer's inference. Thus, the total inference delay equals the sum of each layer's delay. Let $F_i(j,k)$ denote the cumulative computation delay from the 1_{st} to the j_{th} neural network layer for one sample inference of task i on device k where $k \in (I \cup E \cup C)$, and let $G_i(j)$ denote the output size of the j_{th} layer for one sample. Also for task i, c_i^1 and c_i^2 are the two partitioning parameters to represent the first and the second partitioning position, where $c_i^1, c_i^2 \in 0, 1, ..., l_i$.

For ease of exposition, we use $\bar{t}_i^1$, $\bar{t}_i^2$, $\bar{t}_i^3$, $\bar{t}_i^4$ and $\bar{t}_i^5$ to represent the five intermediate delay components for task i's inference. The five delays represent the device computation delay, the data transmission delay over the edge network, the edge server computation delay, the data transmission delay over the backbone network and the cloud computation delay, respectively. We also use $\bar{t}_i^6$ to denote the communication delay for sending the inference result back to the device. These

delays can be calculated as follows:

$$\bar{t}_i^1 = F_i(c_i^1, i) * N_i, \forall i \in I \quad (1)$$

$$\bar{t}_i^2 = \sum_{e \in E} x_{ie} * G_i(c_i^1)/b_i^1 * N_i, \forall i \in I \quad (2)$$

$$\bar{t}_i^3 = \sum_{e \in E} x_{ie} * (F_i(c_i^2, e) - F_i(c_i^1, e)) * N_i, \forall i \in I \quad (3)$$

$$\bar{t}_i^4 = \sum_{e \in E} y_{ie} * G_i(c_i^2)/b_i^2 * N_i, \forall i \in I \quad (4)$$

$$\bar{t}_i^5 = \sum_{e \in E} y_{ie} * (F_i(l_i, c) - F_i(c_i^2, c)) * N_i, \forall i \in I \quad (5)$$

$$\bar{t}_i^6 = \sum_{e \in E} (y_{ie} * G_i(l_i)/b_i^2 + x_{ie} * G_i(l_i)/b_i^1) * N_i, \forall i \in I \quad (6)$$

Our objective is to minimize the total of each job's delay, which can be calculated as below:

$$\min \sum_{i \in I} (\bar{t}_i^1 + \bar{t}_i^2 + \bar{t}_i^3 + \bar{t}_i^4 + \bar{t}_i^5 + \bar{t}_i^6) \quad (7)$$

s.t.

$$\sum_{i \in I} z_{ie} \leq r_e, \forall e \in E \quad (8)$$

$$\sum_{i \in I} b_i^1 * x_{ie} \leq B_e^1, \forall e \in E \quad (9)$$

$$\sum_{i \in I} b_i^2 * y_{ie} \leq B_e^2, \forall e \in E \quad (10)$$

$$0 \leq b_i^v \leq B_i^v, \forall i \in I, v \in \{1, 2\} \quad (11)$$

$$x_{ie}, y_{ie}, z_{ie} \in \{0, 1\}, \forall i \in I, e \in E \quad (12)$$

$$x_{ie}, y_{ie}, z_{ie} = 0, \forall i \in I, \forall e \notin E_i \quad (13)$$

$$0 \leq c_i^1 \leq c_i^2 \leq l_i, \forall i \in I \quad (14)$$

$$y_{ie} \leq x_{ie}, \forall i \in I, e \in E \quad (15)$$

$$z_{ie} \leq x_{ie}, \forall i \in I, e \in E \quad (16)$$

$$\sum_{e \in E} x_{ie} = \begin{cases} 0, c_i^1 = l_i \\ 1, c_i^1 \neq l_i \end{cases} \quad \forall i \in I \quad (17)$$

$$\sum_{e \in E} y_{ie} = \begin{cases} 0, c_i^2 = l_i \\ 1, c_i^2 \neq l_i \end{cases} \quad \forall i \in I \quad (18)$$

$$\sum_{e \in E} z_{ie} = \begin{cases} 0, c_i^1 = c_i^2 \\ 1, c_i^1 \neq c_i^2 \end{cases} \quad \forall i \in I \quad (19)$$

Equation (8) - (11) are the edge server workload constraint, the downlink bandwidth constraint of each edge server, the uplink bandwidth constraint of each edge server, and upper bound constraint of each connection respectively, demanding that the allocation of the computation and bandwidth resources should not exceed the limitation. Equation (14) is the partitioning constraint which requires the first partitioning position in the neural network should not be set behind the second partitioning position. Equation (15) - (19) are the constraints for the three indicator variables.

IV. Partitioning and Scheduling Solution

In the problem formulated in Section III, the variables c_i^1 and c_i^2 which control the partitioning position will influence the variables x_{ie}, y_{ie} and z_{ie} which correspond to the scheduling scheme in Equations (16) - (18), while the scheduling scheme will in turn influence the partitioning strategy defined in Equations (1) - (6). However, if we fix the values of network environment variables, we can calculate the best partitioning position for the shortest inference delay using the simple exhaustive search. To this end, we solve the problem in two steps. We first make a reformulation for the problem after we get the best partitioning scheme for each task under certain network environment, then we use a delayed scheduling strategy to solve the scheduling problem.

A. Problem Reformulation

Because of different partitioning schemes will influence the resource utilization, to circumvent this problem, we first calculate the different optimal delays under different execution environment by using the simple exhaustive search. We use a matrix $W = \{w_{i,j} | i \in I, 0 \leq j \leq 3m\}$ to describe the optimal delays, where $w_{i,0}$ equals to the optimal delay when running the entire model on the mobile device; $w_{ij}, \forall 1 \leq j \leq m$ equals to the optimal delay when using the computation resource on device i and edge server j; $w_{i,j+m}$ equals to the optimal delay when using the computation resource on device i and the cloud c while the communication link is set via edge server j; and $w_{i,j+2m}$ equals to the optimal delay when using the computation resource on device i, edge server j and the cloud c. We also define $q_{i,j} \in \{0, 1\}, i \in I, 0 \leq j \leq 3m$ as the resource utilization indicator , where $q_{i,j} = 1$ means task i is scheduled using the corresponding resource indexed by j, and $q_{i,j} = 0$ otherwise.

We can thus reformulate our problem (1)-(18) as below:

$$\min \sum_{i \in I} \sum_{0 \leq j \leq 3m} w_{i,j} * q_{i,j} \quad (20)$$

s.t.

$$\sum_{i \in I} q_{i,j} + q_{i,j+2m} \leq r_{e_j}, \forall 1 \leq j \leq m \quad (21)$$

$$\sum_{i \in I} b_i^1 * (q_{i,j} + q_{i,j+m} + q_{i,j+2m}) \leq B_j^1, \forall 1 \leq j \leq m \quad (22)$$

$$\sum_{i \in I} b_i^2 * (q_{i,j+m} + q_{i,j+2m}) \leq B_j^2, \forall 1 \leq j \leq m \quad (23)$$

$$\sum_{0 \leq j \leq 3m} q_{i,j} = 1, \forall i \in I \quad (24)$$

$$q_{i,j} \in \{0, 1\}, \forall i \in I, 0 \leq j \leq 3m \quad (25)$$

and (11).

The constraints (21) - (23) are equivalent to constraints (8) - (10). Constraint (24) and (25) restrict that for each task, the scheduler must provide exactly one execution scenario.

B. Online Scheduling for the Multi-task Execution

In this subsection, we will propose a scheduling solution for the multi-task execution. Under the real-world environment,

tasks are usually not generated at the same time. Therefore, we design an online scheduling algorithm to better handle this situation, as illustrated in Algorithm 1.

In Algorithm 1, we propose a new concept, event, which can be defined as a scheduled task's starting or ending, or an unscheduled task's coming. Triggering an event is the only way to affect the system's resource usage. When the triggered event is the start or the end of a scheduled task, the algorithm will update the current set of tasks (Line 2 - 3). When the event is a new task's coming, Algorithm 1 will arrange the edge network connection and allocate the bandwidth resource for the task. For the bandwidth allocation, since the optimized objective is the delay, we allocate the bandwidth as large as possible (Line 11). The highest allocable bandwidth between the device i and the edge server j is decided by B_i^1, as well as the current allocable downlink bandwidth of edge server j. It can be calculated as $B_j^1 - \sum_{i \in I} b_i^1 * x_{ij}$. The highest allocable bandwidth between the edge server j and the cloud c can be calculated similarly.

Algorithm 1 also adopts the delayed scheduling strategy. For a specific task, it is possibly to achieve lower delay when queuing it for the incoming freed resource by other tasks. When adopting the delayed scheduling, we enumerate each of the coming events. We will choose to delay a task's starting if total delay can be reduced (Line 12 - 17).

To analyze the complexity our algorithm, we define $n_{current}$ as the current number of the concurrent executing tasks. The enumeration of events will take $O(n_{current})$ time since each task will trigger the event when it starts or ends after scheduled. And in each enumeration, to calculate the current resource capacity of each edge server will take $O(n_{current} * m)$ time. Thus, the time complexity of Algorithm 1 is $O({n_{current}}^2 * m)$ for each coming task and $O(n * {n_{current}}^2 * m)$ for all n tasks. Note $n_{current}$ is constrained by the limited resource on the edge servers, so it is far smaller than n.

Algorithm 1: Online Scheduling

Data: i;
m_range = set of integers in $[0, 3m]$;
$w_{i,j}, \forall j \in m\_range$;
Result: opt, opt_{delay};
$q_{i,j}, \forall j \in m\_range$

```
1  while there is an event et happens do
2      if et is task i's start(end) then
3          Add(remove) task i in set I;
4      if et is a new task i coming then
5          opt = +infinty;
6          for each known event et_known happens not
             before et do
7              Temporarily update the task set I for
                 et_known;
8              Calculate the time interval between et and
                 et_known as δt;
9              for ∀j ∈ m_range do
10                 if arrange q_{i,j} doesn't violate current
                     resource restriction then
11                     Allocate the maximum allocable
                         bandwidth for b_i^1 and b_i^2;
12                     if w_{i,j} + δt < opt then
13                         opt_delay = δt;
14                         opt = w_{i,j} + δt;
15                         temp = j;
16                     else
17                         Reset b_i^1 and b_i^2 with value 0;
18 return opt_delay, opt, q_{i,temp} = 1, b_i^1 and b_i^2;
```

V. Performance Evaluation

We conduct prototype experiments and extensive simulations to evaluate our framework and solutions. In particular, we first use real-world prototype experiments to gather execution information about the DNN inference delay for each layer on each platform, which further verifies the practical usefulness of our framework. Based on the gathered information, we then conduct extensive simulations to examine the performance of our solutions proposed in Section. IV.

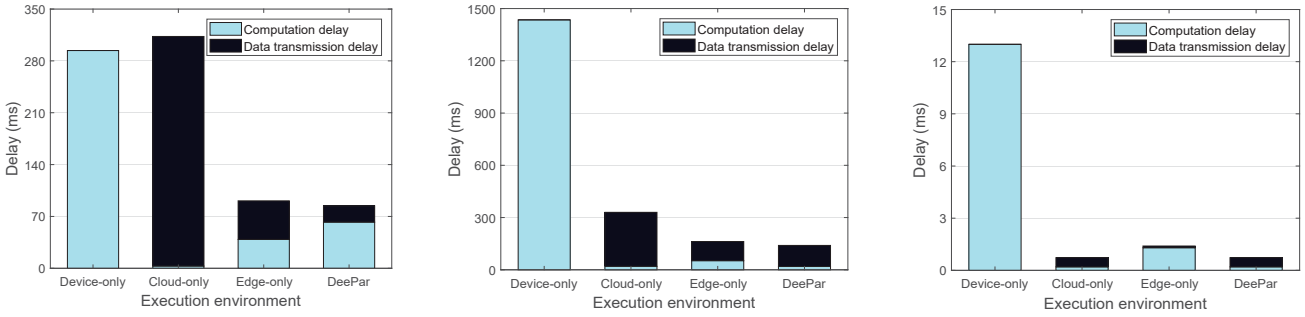
A. Real-world Prototype Experiments for Delay Reduction

In our experiments, we implement a simple prototype and examine more state-of-the-art deep neural networks such as VGG [8], DeepFace [1] and LeNet [9], to better investigate the effectiveness of our DeePar framework. The mobile device is a Google Pixel XL mobile phone with a Qualcomm Snapdragon 835 CPU. The edge server is a Dell server (OPTIPLEX 7010) and equipped with an NVIDIA GeForce GTX 1080 Ti GPU, an Intel Core i7-3770 3.4 GHz quad core CPU and a 16 GB 1333 MHz DDR3 RAM. The cloud platform is the Google Cloud Platform with five instances with each equipped with a 13 GB RAM and a NVIDIA Telsa K80 GPU. The mobile device connects to the edge server through a Wi-Fi environment. The programming backend is Tensorflow [10]. For VGG and DeepFace, the used dataset is a set of pictures with each of the size around 200KB, while for LeNet, the dataset is the MNIST dataset with each picture around 1KB.

The experiment result is shown in Fig. 3. Our DeePar outperforms all other approaches for VGG and DeepFace, with overall 10% -30% delay reduction comparing to the edge-only execution. The result for LeNet is quite different, since the data size is very small here, data communication delay is not the largest overhead here. In this case, our DeePar achieves the same delay as the cloud-only execution, which is much lower than the device or the edge-only execution, demonstrating that edge computing is not always effective to reduce the delay.

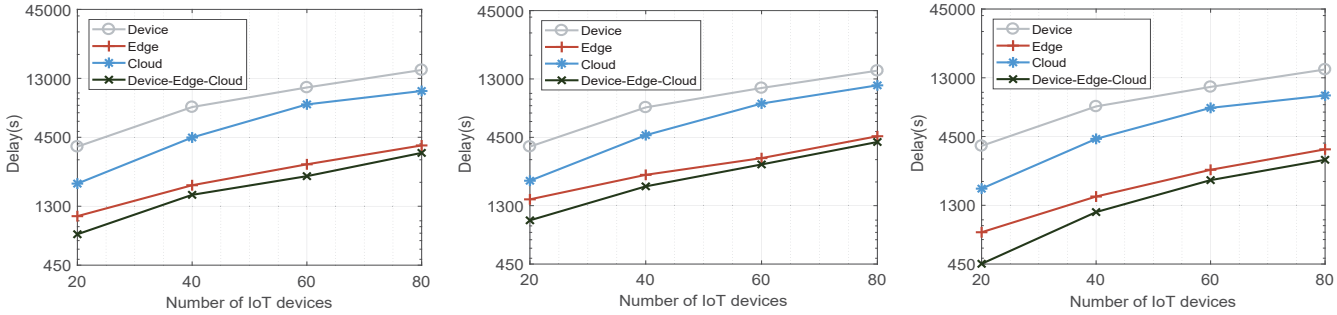
B. Simulations for Multi-task Scheduling

Based on the data gathered in the above real-world experiments, we further conduct extensive simulation tests to examine the performance of our solutions for multi-task execution, where the tasks are mixed with all the four types of DNNs we have examined previously, namely, AlexNet, VGG, DeepFace and LeNet. Specifically, B_i^1 is set between [20, 300] Mbps



(a) VGG (b) DeepFace (c) LeNet

Fig. 3: Inference delay of various DNNs by real-world prototype experiments



(a) Time interval within 300s (b) Time interval within 50s (c) With 40 edge servers

Fig. 4: Inference delay of various DNNs by simulations for multi-task scheduling

and B_i^2 is set between [10, 50] Mbps, where $\forall i \in I$; B_e^1 is set in [5, 20] Gbps, B_e^2 is set in [0.1, 5] Gbps, and r_e is set in [1, 10], where $\forall e \in E$; N_i is set in [1, 1000] and each input's size ($G_i(0)$) is set in [10, 1000] KB, where $\forall i \in I$. The above used notations are defined in Section III.

We first examine how our solutions perform with 10 edge servers, where the number of devices changes from 20 to 80, and all tasks arrive one by one within a time interval of 300 seconds. The result is shown in Fig. 4(a). We can see that DeePar achieves over 60% delay reduction compared to the device-only or the cloud-only execution, and about 20% delay reduction compared to the edge-only execution. When we shorten the time intervals for each task within 50 seconds, as shown in Fig. 4(b), DeePar can still reduce the total inference delay for about 70% comparing to the cloud-only execution and up to 30% comparing to the edge-only execution. We then increase the number of edge servers to 40. As shown in Fig. 4(c), DeePar can still achieve the best performance with about 80% delay reduction compared to the device-only and 40% delay reduction compared to the edge-only execution.

VI. Conclusion

In this paper, we proposed the DeePar framework which explores edge computing to reduce data communication delay and adopts a layer-level partitioning strategy to further improve the overall inference performance. We also designed an efficient heuristic online algorithm to solve the multi-task execution problem. Through real-world prototype experiments and simulations, we showed that our DeePar solution can achieve the best performance among all approaches with up to 80% reduction in total inference delay.

VII. Acknowledgments

This research is supported by a Canada NSERC Discovery Grant and an Engage Grant.

References

[1] O. M. Parkhi, A. Vedaldi, A. Zisserman *et al.*, "Deep face recognition." in *British Machine Vision Conference (BMVC)*, 2015.

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[4] 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 *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[5] H. Li, K. Ota, and M. Dong, "Learning iot in edge: deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[7] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2018.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[9] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.