DNN Partitioning and Inference Task Offloading in 6G Resource-Constrained Networks

Dimitrios Kafetzis and Iordanis Koutsopoulos
Department of Informatics, Athens University of Economics and Business, Greece

***Abstract*—In the emerging 6G networks landscape, edge computing applications are tasked with performing Machine-Learning (ML) inference using Deep Neural Networks (DNNs) on resource-constrained devices in terms of computational power, memory, and energy. This paradigm shift, brought forth by 6G's promise of ultra-reliable low-latency communication necessitates novel approaches for managing DNN tasks. This paper studies the problem of DNN partitioning and selective offloading of ML inference tasks to a Base Station (BS) or an edge gateway node equipped with substantial computational resources. We consider a set of resource-constrained devices, each of which has an inference task (abstracted as a DNN) to execute. DNN partitioning determines the part of the neural network to run locally, and the part to offload to the BS. We are interested to determine a suitable task partition for each device, namely determine the neural network layer up to which computation will take place locally on the device. A certain task partition on a device affects the inference delays of tasks of other devices due to task coupling, when task scheduling for processing is decided at the BS. We formulate the optimization problems of DNN partitioning for minimizing total execution delays of tasks and for providing fair treatment to DNN inference tasks, by ensuring balanced task execution delays. We propose a greedy heuristic algorithm to solve the problems, and we evaluate it through numerical simulations, using input from experiment measurements on Raspberry Pi devices. The proposed approach is shown to perform much better than the baseline approach where each task is entirely executed locally on the device.**

***Index Terms*—DNN partitioning, Task offloading, Resource-constrained devices, Low-latency communication.**

I. Introduction

With the advent of 6G networks, Edge Computing and the Internet of Things (IoT) undergo transformative changes. 6G is anticipated to enhance connectivity, reduce latency, and provide higher bandwidth, thereby supporting the proliferation of connected pervasive systems. These systems will support a wide array of applications and services by processing vast amounts of data generated by sensors and other devices at an even higher rate. In such an ecosystem, swarms of mobile, wearable, and other lightweight devices will play a crucial role by executing complex inference tasks. However, these devices are often limited by computational capacity, memory, and energy resources.

An integral part of envisioned 6G services is the ability to deliver the results of ML inference such as prediction or classification in a timely manner. Deep Neural Networks (DNNs), required for executing inference tasks, consist of multiple layers and stages, such as convolutional and pooling layers which demand significant compute resources [1]. The challenge of how to execute these tasks optimally in terms of inference latency on resource-constrained devices within the 6G context is both critical and complex.

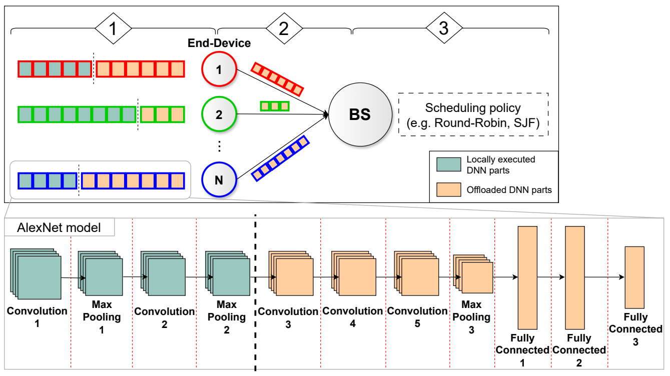


Fig. 1: (**Top part**) The 3 stages of the proposed DNN partitioning approach: 1) Decision on DNN partitioning, 2) Local execution of the first DNN part and transmission of intermediate results to the Base Station (BS), and 3) Execution of the offloaded DNN part at the BS. (**Bottom part**) The AlexNet model, exemplifying a DNN model for inference tasks.

The prevalent approach to address the challenge above is to use a powerful edge server, such as a Base Station (BS). The BS receives requests for inference tasks from devices, it executes the inference tasks, and it returns the results back to the devices. However, this approach leads to large incurred delays which may jeopardize the stringent low-latency specifications posed by real-time inference response requirements of the service for which the DNN inference task is executed.

In this work, we advocate another option, depicted in Fig. 1. This involves three stages: (1) the decision to partition the task (i.e. the chain of DNN computations of the neural network) into two parts, one executed locally at the device, and the other executed at the BS, (2) the transmission to the BS of the intermediate result after the local computation so that the BS executes the second part of the chain, and (3) the execution of this second part of the task at the BS.

In Fig. 1 (top), we depict a set of devices, each of which needs to execute a multi-layer DNN. The task inference delay

This work has been carried out in the context of the research project entitled "Wireless Mobile Delay-Tolerant Network Analysis and Experimentation, Project acronym: LEMONADE" (Project Number: HFRI-FM17-352) that was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "1st Call for H.F.R.I. Research Projects to support Faculty Members & Researchers and the Procurement of high-cost research equipment grant".

comprises three components: *(i)* the execution delay of the part of the chain that is executed locally at the device; *(ii)* the delay for transmitting intermediate results to the BS via a wireless channel, and *(iii)* the execution delay of the rest of the DNN at the BS processor. The third delay component above implies that the decision of DNN partitioning for a certain device affects the execution delays of all other devices as well. This occurs because the task portions from all devices that are offloaded to the BS are scheduled based on a given scheduling policy at the BS such as Round-Robin (RR) or Shortest-Job-First (SJF). Hence, depending on the scheduling policy at the BS, the size of a computation task that is served at the BS, affects the delays of other tasks. It is this coupling at the BS that makes the problem challenging and nontrivial to solve, because the DNN partitioning decisions need to be taken *jointly* for all devices, rather than separately for each one of them.

Assuming without loss of generality that there exists one inference task to be executed for each device, the main question is to find a DNN partitioning policy that minimizes the total inference delay of all tasks. This is a first natural objective we consider in this paper. However, such a policy might turn out to cause large inference delays for some devices since the minimization of sum delay does not imply the minimization of each delay separately. Thus, we next ask the question: *how should we partition the DNNs of the different devices so as to attain an execution delay that is as balanced as possible across devices?* The objective of balancing DNN execution delays across devices stems from the requirement to treat inference tasks at all devices fairly, in light of real-time inference requirements in all of them. A solution that caters for balancing of DNN execution delays across devices provides a delay that is as equalized as possible across devices. To the best of our knowledge, this is the first work that addresses such optimization problems concerning the DNN partitioning decision, while taking into account the aspect of coupling of the partition decisions of the different DNN tasks at a BS.

A. Our contributions

This paper makes the following contributions:

- We formulate the optimal DNN partitioning problem as an optimization problem, and we aim to find the partitioning for each end-device in order to (*i*) minimize total execution delay, and (*ii*) minimize the variance of total execution delays of DNN tasks.
- We take into account in the model the range of feasible partitions for each device that arise due to realistic constraints on computation capacity and memory resource availability, as well as the impact of parallel background processes onto DNN task execution on end-devices.
- We provide a low-complexity greedy DNN task partitioning algorithm to solve the problem that overcomes its computational complexity.
- We use real experimental measurements from DNN execution on Raspberry Pi devices with specific resource limitations and transmit capabilities to inform our mathematical model and feed the simulation model. We compare the performance of the partitioning algorithm to that where each DNN task is executed locally on the device, and we show that our method significantly reduces DNN task execution delay variance.

The rest of the paper is organized as follows. Section II reviews previous research. In section III, we present the system model and problem statement. The evaluation approach, evaluation setup and results are discussed in section IV. Section V concludes the paper.

II. Related Work

Some works in the literature (e.g. [2]) focus on distributed training of DNN models, and they propose partitioning methods to accelerate training. Other works e.g. [7] and [8] consider only the full offloading of DNN inference tasks, and their objective is either the minimization of energy consumption or the minimization of the amount of circulated network traffic. Instead, in this work we focus on the DNN partitioning at the *inference* stage, for selective partial offloading of DNN inference tasks to a BS.

Both DNN partitioning and offloading have been considered as separate problems in some works, e.g. [3] and some others consider jointly partitioning and offloading of multiple DNNs using solvers like SPSO-GA [4], combining Particle Swarm Optimization (PSO) and Genetic Algorithm (GA), and DDPQN [5], which employs Deep Reinforcement Learning for determining partitioning schemes. Instead to our approach these works do not consider the coupling of the offloaded DNN partitions at a BS and the scheduling process there.

Another class of works focus on DNN execution for IoT application scenarios. A work that considers constrained devices is [6], which focuses on the problem of distributing deep-learning (DL) processing on different, potentially heterogeneous IoT devices. Their method assigns DNN layers to IoT devices to minimize delay, considering each device's memory and computational constraints. Our approach combines DNN partitioning with scheduling of DNN tasks at the BS side to minimize inference delays and ensure fair task distribution across devices considering their resource capabilities.

The method we study is conceptually similar to that of Split Learning [10], [11] where a NN is partitioned during training, and different parts of it are executed at different locations (devices or a central server). However, our work differs from the line of works in Split Learning since (*i*) it is concerned with inference rather than training, (*ii*) it addresses the part of computation scheduling at the BS, and (*iii*) it introduces a solid mathematical optimization framework for *jointly* optimizing total delay or delay variance across different learning tasks.

Closer to our approach are the works [12] and [13]. In these works, the authors study the DNN partitioning and offloading problem for tasks that can be modeled as a directed acyclic graph. Each link of the graph corresponds to a subtask, and the problem is to offload subtasks to devices so as to minimize the makespan of all tasks, while respecting the deadlines, given different task release times and deadlines.

In our work, we formulate the problem of DNN task partitioning and offloading from a set of devices to a BS, where each device generates a distinct task. This problem is challenging since the decisions of DNN partitioning of different devices need to be taken jointly, and they are influenced by the scheduling policy at the BS. We also formulate the problem of total task delay minimization and that of delay balancing, through optimizing delay variance.

III. Model and problem formulation

A. Inference task

We consider the snapshot version of the following setup. There exist N devices, each of which needs to execute a ML inference task. We assume that an ML model (e.g. a DNN) has already been trained and has been loaded on the device. When a request for inference based on the ML model arises, it means that a feature vector (e.g. an image) is presented to the device. That vector is inserted as input to the DNN and traverses layer-by-layer the chain of DNN layers that the device executes so as to compute the output of the DNN, which can be for example the predicted label (or class) of the feature vector.

In our model, each device may execute a portion of the DNN task (i.e. some neural network layers) locally, and offload the rest to the BS. The BS receives the offloaded portions of DNN tasks by the end-devices and executes them according to a certain scheduling policy, to be detailed below. We assume that the DNN model has also been pre-loaded at the BS.

The DNN task can thus be described as a sequence of stages (layers) $\mathcal{S} = \{1, \dots, S\}$. Each stage has as input the resulting weights of the previously computed stage e.g., stage 2 has to be executed after stage 1. For example, in the AlexNet DNN model [9] of Fig. 1, each layer (or group of layers, e.g. convolution 1, max pooling 1, etc) is a stage $s \in \mathcal{S}$ of the inference task, and its computational requirements are measured in floating point operations (flops). For a device i, the partition point p_i of the DNN can be placed at any stage $s = 1, \dots, S$. In Fig. 1 we indicate the possible partitioning points of AlexNet model. In reality, memory constraints at the device impose limitations on the portion of computation that can be executed locally. Hence, for each device i, there exists a maximum number of stages $\tilde{s}_i \leq S$ that can be executed locally.

B. Inference delay

There exist three main components of the total inference delay for the DNN task of each device i:

1) Local computation delay, D_i^L, which is the execution delay for part (or the entire) chain of layers that are executed on the end-device i.
2) Transmission delay, D_i^{tr}, which is time needed so that the device transmits to the BS either the output weights of the last locally executed DNN stage or the input feature vector (e.g. image). In the latter case, no local execution occurs, and the task is fully offloaded to the BS.
3) BS (remote) computation delay, D_i^R, which is the time needed so that the residual of each DNN task is executed at the BS server.

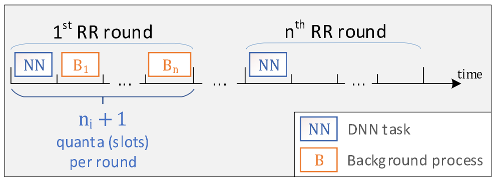


Fig. 2: The RR scheduling of the DNN process (task) and background tasks at an end-device processor.

1) Local computation delay: The end-device has a processor whose computational capacity is C_i flops/sec, and it is shared among the inference task and possibly other tasks that are beyond our control, such as certain background processes. We assume that a default RR scheduling discipline runs at each end-device to manage execution of the DNN task and other background tasks. RR is a preemptive scheduling algorithm, where each task is periodically assigned a fixed time slot of duration q to execute. These slots realize time sharing of the computational capacity of the device. When a time slot expires, the execution of the next task occurs for a duration q, and so on, as depicted in Fig. 2.

We fix attention to a snapshot version of the problem, where there exists one DNN task to be executed at each device i, together with n_i background tasks that are continuously backlogged during DNN task execution, i.e. if a background process ends, another comes up. Local computation at device i takes a total amount of time $F_i^L(p_i)/C_i$, where $F_i^L(p_i)$ is the number of flops to be executed at the device, and $p_i \in \{1, \dots \tilde{s}_i\}$ is the selected partition point.

The duration of a RR round is thus $(n_i + 1)q$. By realizing the RR sequence of tasks, we can find that the time needed until the DNN task is executed with RR scheduling at device i for partition point p_i is

$$D_i^L(p_i) = q\left(\frac{F_i^L(p_i)}{C_i}(n_i + 1) - n_i\right). \tag{1}$$

2) Transmission delay: Each device i is connected to the BS with a wireless channel of average transmission rate W_i bits/sec. Let $A_i(p_i)$ be the number of bits that are transmitted from device i to the BS when partition point p_i is chosen. The transmission delay is $D_i^{tr}(p_i) = A_i(p_i)/W_i$.

3) Base station computation delay: We assume that the BS is equipped with a processor of computational capacity C^R flops/sec. A certain scheduling policy $\mathcal{G}$ is employed at the BS to serve the residual DNN tasks from all devices that are offloaded to the BS. We assume there are no background processes at the BS.

The size (namely, the number of flops) of the DNN task of device i that needs to be executed at BS when the partition point is p_i, is $F_i^R(p_i) = F_i - F_i^L(p_i)$, where F_i is the total size of DNN task of device i in flops. The total time needed to execute the part of the DNN task of device i at the BS, when partition p_i is selected, is $F_i^R(p_i)/C^R$. Note that this time would be required if that task were the only one to be executed at the BS.

Let $\mathbf{p} = (p_1, \dots, p_N)$ be the sought partition policy, i.e. the vector of partition points of each device. For a certain scheduling policy $\mathcal{G}$, let $D_i^R(\mathbf{p}; \mathcal{G})$ denote the total time that is required for executing at the BS the portion of the DNN task of device i which is offloaded there.

Observe that for a certain scheduling policy, the delay above for a certain device i depends not only on partition point p_i, but on the entire partition policy $\mathbf{p}$. This is because the execution delay for the task of a certain device that is sent to the BS does not depend only on its own task size, but also on the sizes of tasks of other devices that are sent to the BS, each of which is determined by the partition point of these devices.

Example: Shortest Job First (SJF) scheduling. If the scheduling policy at the BS is the SJF one, the sizes of DNN tasks that are sent to the BS are ranked in increasing order of execution delays at the BS, and they are executed at the BS in that order. In fact, depending on the sizes of the residual tasks from all devices that are sent to the BS, there exists a natural ranking on the sizes of these tasks at the BS that is used for scheduling.

For a certain partition policy $\mathbf{p}$, let $\beta_i(\mathbf{p})$ be the rank of the task of device i. For example, $\beta_i(\mathbf{p}) = 1$ means that $F_i^R(p_i) = \min_j F_j^R(p_j)$, i.e. the task of device i ordered first, i.e. it is the shortest task from those sent to the BS; if $\beta_i(\mathbf{p}) = N$, this means that device i is ordered last, i.e. it has the longest task sent to the BS. Again, this ordering depends on the choice of the partition.

The computation delay of the DNN task of device i at the BS depends on the rank $\beta_i(\mathbf{p})$ of that task and is given by

$$D_i^R(\mathbf{p}; SJF) = \frac{1}{C^R}\left(F_i^R(p_i) + \sum_{j:\beta_j(\mathbf{p})<\beta_i(\mathbf{p})} F_j^R(p_j) \right). \quad (2)$$

Depending on the scheduling policy $\mathcal{G}$, we may find an analytical expression for the execution delay $D_i^R(\mathbf{p}; \mathcal{G})$. The total inference delay for device i is

$$D_i(\mathbf{p}; \mathcal{G}) = D_i^L(p_i) + D_i^{tr}(p_i) + D_i^R(\mathbf{p}; \mathcal{G}) \, . \quad (3)$$

C. Problem formulation

Due to real-time DNN inference response delay requirements, there exists a maximum allowable application-driven inference delay $\tilde{D}_i^{max}$. In the sequel, we omit the notation $\mathcal{G}$ for simplicity. We consider two performance objectives and formulate the respective optimization problems: (*i*) total inference delay minimization, and (*ii*) inference delay variance optimization.

1) Total inference delay minimization: The total inference delay for DNN tasks of all devices is

$$D_{tot}(\mathbf{p}) = \sum_{i=1}^{N} D_i(\mathbf{p}) \, . \quad (4)$$

A natural objective would be to seek the DNN partition policy $\mathbf{p}$ that minimizes total inference delay, $D_{tot}(\mathbf{p})$ above. This problem is mathematically formulated as follows:

$$\begin{aligned} \min_{\mathbf{p}} \quad & D_{tot}(\mathbf{p}) \\ \text{s.t.} \quad & D_i(\mathbf{p}) \le \tilde{D}_i^{max} \ \forall \text{ device } i = 1, \dots, N \\ & p_i \in \{1, ..., \tilde{s}_i\}, \end{aligned} \quad (5)$$

where $D_i(\mathbf{p})$ is given by (3).

The problem is computationally hard to solve because of the integer solution space. One would need to consider all possible combinations of partitions for each device, compute inference delays and delay variance, and then choose the partition that minimizes this variance. We stress that the complexity stems from the fact that the partitions of different devices need to be computed *jointly* since one partition in general affects the delays of all other devices due to their coupling through the BS scheduling policy. For N devices and S partition points, there are S^N possible solutions. While exhaustive enumeration might be an option for very few devices and simple neural networks with few stages, having some tens of devices and deep neural networks would render the solution space very large.

We now describe a greedy heuristic algorithm to solve the problem. We start from an arbitrary initial partition $\mathbf{p}^{(0)}$, e.g. $\mathbf{p}^{(0)} = (\lfloor \frac{\tilde{s}_1}{2} \rfloor, \dots, \lfloor \frac{\tilde{s}_N}{2} \rceil)$ or $\mathbf{p}^{(0)} = (1, 1, \dots, 1)$. The idea is at each step to change the partition of the task of one device such that the largest reduction in total delay occurs.

At each iteration $t \ge 1$, given the partition $\mathbf{p}^{(t-1)} = (p_1^{(t-1)}, \dots, p_N^{(t-1)})$, and a total delay value $D_{tot}(\mathbf{p}^{(t-1)})$ from the previous step, for each device i we define the following two tentative partitions: (*i*) a partition $\mathbf{p}^{i+}$, in which we move the partition point of device i task one position to the right, while all other device task partitions are unchanged; and (*ii*) a partition $\mathbf{p}^{i-}$, in which we move the partition point of device i task one position to the left, and all other device task partitions are unchanged. Let $\Delta D^{i+} = D_{tot}(\mathbf{p}^{(t-1)}) - D_{tot}(\mathbf{p}^{i+})$, and $\Delta D^{i-} = D_{tot}(\mathbf{p}^{(t-1)}) - D_{tot}(\mathbf{p}^{i-})$. Thus, we have $2N$ possible partitions.

We choose to move the partition of the task of device i^* with

$$i^* = \arg \max_{j=1,\dots,N} \max\{\Delta D^{j+}, \Delta D^{j-}\} \quad (6)$$

for which $\max\{\Delta D^{j+}, \Delta D^{j-}\} > 0$. Depending on whether ΔD^{i^*+} or ΔD^{i^*-} is larger, we choose partition $\mathbf{p}^{i^*+}$ or $\mathbf{p}^{i^*-}$, namely we move the partition of device i^* task one position to the right or one position to the left, if possible. Then, we set the current partition to $\mathbf{p}^{(t)} = \mathbf{p}^{i^*+}$ or $\mathbf{p}^{(t)} = \mathbf{p}^{i^*-}$, and we continue with the next iteration $t+1$. The algorithm terminates if at some iteration, it is $\Delta D^{i+} < 0$ and $\Delta D^{i-} < 0$ for all devices i, namely it is not possible to further improve the total delay.

2) Inference delay variance minimization: A partition policy that minimizes total inference delay $D_{tot}(\mathbf{p})$ may result in very different values of inference delays for some devices, and it may happen that some of them fail to fulfil maximum delay requirements. Placing delay requirements as constraints in a sum inference delay minimization problem would constrain the space of feasible solutions and would still result in solutions with unequal inference delays.

Another possibility would be to seek to treat devices equally in terms of inference delay guarantees, hence we seek to find the DNN partition policy $\mathbf{p}$ that minimizes the total variance, $V(\mathbf{p})$ of inference delays of DNN tasks of devices, in order to cater for most balanced inference delays across devices. This problem is mathematically formulated as follows:

$$\begin{aligned} \min_{\mathbf{p}} \quad & V(\mathbf{p}) \\ \text{s.t.} \quad & D_i(\mathbf{p}) \leq \tilde{D}_i^{max} \ \forall \text{ device } i = 1, \ldots, N \\ & p_i \in \{1, ..., \tilde{s}_i\}, \end{aligned} \tag{7}$$

where $V(\mathbf{p})$ is given by the following equation:

$$V(\mathbf{p}) = \frac{1}{N} \sum_{i=1}^{N} (D_i(\mathbf{p}) - \bar{D}(\mathbf{p}))^2, \tag{8}$$

where $\bar{D}(\mathbf{p}) = \frac{1}{N} \sum_{i=1}^{N} D_i(\mathbf{p})$ is the empirical mean of individual device inference delays. A greedy algorithm similar in spirit to the one described above can be used for this problem as well.

IV. Evaluation

A. Evaluation setup

The evaluation of our method for DNN partitioning is based on 5 scenarios, where 10, 20, 30, 40, and 50 end-devices are used. Devices are located in various positions, with distance a range of 50 up to 500 meters from the BS. Devices differ in terms of memory and number of background processes that run in parallel to the DNN task. As a result, $\tilde{s}_i$, i.e. the maximum number of stages that can be executed locally is different for each device i. For example, the memory of some devices can support the execution of DNN tasks up to stage 5, while in some other up to stage 8.

Experiments were carried out with a custom *Python*-based network simulator that we designed and developed. The area of the simulated network is a square grid of 1000 m^2. We used real benchmark results of the execution of an AlexNet's *Python* implementation on top of *Tensorflow* and *Keras* framework, a *Python* interface for *Tensorflow*. Our model in the simulator is based on real device measurements. To assess the inference delay of AlexNet's stages, we started with the first layer to measure its execution delay, then sequentially added and measured each subsequent layer of the DNN until all stages were included. We measured the execution delay for each DNN stage using a 227x227 pixel color image as input to AlexNet, and then calculated the total real inference delays for all DNN partitioning scenarios. In Fig. 4 are depicted the inference delay measurements which took place both at the Intel i7 6700HQ that has the role of the BS processor and at the Raspberry Pi 3B+ device. Transmission delay calculations accounted for data sent from one Raspberry Pi 3B+ with a CYW43455 WiFi module at 2.4GHz to the BS, considering varying data rates to represent different distances between the end device and the BS. These calculations incorporated in the model of our custom simulator.

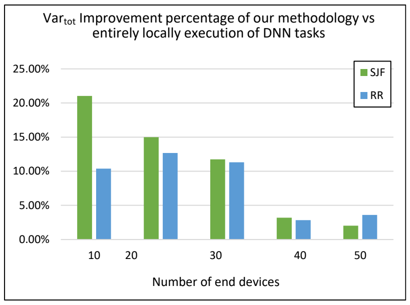
All devices have uniform computational capabilities but differ in memory availability and the number of local parallel processes running alongside the main DNN tasks. Let $R = C^R/C_i$ be the ratio of BS and device i computational capacity. We have executed the evaluation scenarios for 7 different ratios (i.e. $R = \{1, 5, 10, 50, 100, 500, 1000\}$ considering the computational capacity C_i of Raspberry Pi 3B+ as device i, and the computational capacity C^R of an Intel Core i7 6700HQ as the processor of the BS. Using a testbed of these devices, we measured DNN inference delays of each stage under RR and SJF scheduling at the BS. The proposed model and approach are generalizable, allowing each device to run various types of DNNs.

B. Evaluation Results

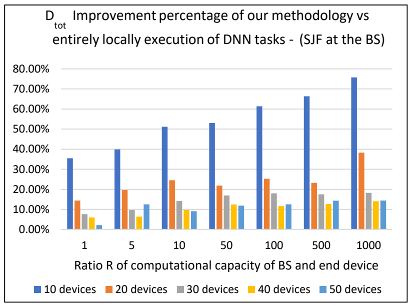
We evaluate our approach in terms of variance of inference delays, and in terms of the total inference delay D_{tot}. We compare our approach to the approach where each DNN is executed entirely on the device. We call the latter option "all-local". Comparative results are depicted in Fig. 3 in the form of percentage of improvement that the proposed approach achieves over the "all-local" one. We also compared the greedy algorithm's results with the optimal solution for minimizing inference delay variance, achieved through exhaustive enumeration of all possible partition combinations. The comparative results are depicted in Fig. 5. The greedy algorithm achieves variance very close to that obtained with the optimal solution, i.e. within 2.4%, for networks of 3 to 10 devices.

With our approach, variance of inference delays is significantly reduced as shown in Figs. 3(A) and 3(C). In particular, in Fig. 3(A), for $R = 1$ and for RR or SJF scheduling at the BS, our proposed methodology performs better than the "all-local" one, with $2\% - 21\%$ smaller variance with SJF scheduling, and $2.8\% - 12.7\%$ smaller variance for RR scheduling, depending on the number of devices $(10, 20, ..50)$. The amount of improvement seems to reduce as the number of devices increases. The proposed method performs better than the "all-local" one for 10-30 devices, but even with more devices (e.g 40 or 50), it still shows better performance in comparison to the "all-local" approach.

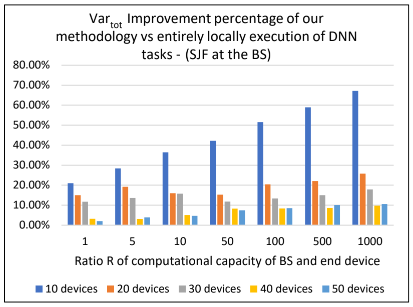
In Fig. 3(C), the improvement in variance is depicted for various values of ratios R ranging from 1 to 1,000. The improvement increases as the BS becomes more powerful, and the largest improvements (ranging from 21% to 67.2%) are observed for 10 devices. Similarly, the proposed methodology gives slightly better results for the reduction of the total inference delay, as depicted in Fig. 3(B). Also, as depicted in Figs. 3(A) and 3(B), the resulting partitions for 10, 20, 30, 40 and 50 devices and $R = 1$ show that the average percentage of locally executed DNN layers increases as the number of devices increases. We note that irrespective of the BS's scheduling policy, the optimal number of DNN layers for local execution rises with an increase in device count.



(A)



(B)



(C)

Fig. 3: Percentage of improvement in the variance of inference delays with our approach, over fully local task execution.

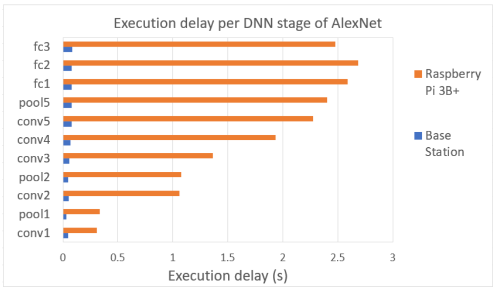


Fig. 4: Inference delay measurements of AlexNet DNN stages on Raspberry Pi 3B+ and Intel i7 6700HQ (the BS processor).

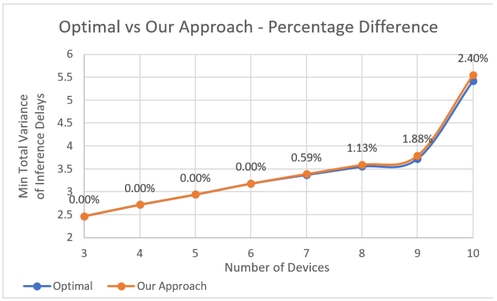


Fig. 5: Optimal variance of inference delays with exhaustive enumeration of possible partitions, vs. variance obtained with the greedy algorithm, for different number of devices. The labels on each point indicate the percentage difference between our algorithm and the optimal solution. The greedy algorithm in almost all cases finds the optimal solution.

V. Conclusion

We addressed the problem of minimizing total inference task delay, as well as that of minimizing the variance of delays, through DNN partitioning and offloading to a BS, for given BS scheduling policies. Task scheduling at the BS makes the decision variables coupled. Our evaluation demonstrates that the proposed DNN partitioning method yields a significant reduction in inference delay variances compared to “all-local” execution, particularly when BS computational power is higher. Also, the results reveal that the efficiency of our method is maintained across different DNN sizes. Despite diminishing gains with as the number of devices increases, our method still consistently surpasses “all-local” execution in all tests, up to the largest evaluated network size of 50 devices. As future steps, we plan to extend our method to also include the minimization of energy consumption of DNN tasks, and to fully implement our proposed methodology on a real hardware testbed.

References

[1] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, “An Early Resource Characterization of Deep Learning on Wearables, Smartphones and Internet-of-Things Devices.,” in IoT-App@SenSys, pp. 7–12, 2015.

[2] A. E. Eshratifar, M. S. Abrishami and M. Pedram, “JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services,” in IEEE Trans. on Mob. Comput., vol. 20, no. 2, pp. 565-576, 2021.

[3] T. Mohammed, C. Joe-Wong, R. Babbar and M. D. Francesco, “Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading,” Proc. of IEEE INFOCOM 2020, pp. 854-863, 2020.

[4] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter and G. Min, “Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments,” IEEE Trans. Parallel Distrib. Syst., vol. 33, no. 3, pp. 683–697, Mar. 2022.

[5] M. Xue, H. Wu, G. Peng and K. Wolter, “DDPQN: An efficient DNN offloading strategy in local-edge-cloud collaborative environments,” IEEE Trans. Serv. Comput., vol. 15, no. 2, pp. 640–655, Mar./Apr. 2022.

[6] S. Disabato, M. Roveri and C. Alippi, “Distributed Deep Convolutional Neural Networks for the Internet-of-Things,” in IEEE Transactions on Computers, vol. 70, no. 8, pp. 1239-1252, 2021.

[7] 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.

[8] S. Chinchali, A. Sharma, J. Harrison, A. Elhafsi, D. Kang, E. Pergament, E. Cidon, S. Katti, M. Pavone, “Network Offloading Policies for Cloud Robotics: A Learning-Based Approach,” in Autonomous Robots, 2021.

[9] A. Krizhevsky, I. Sutskever, G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Commun. ACM, vol. 60, pp.84–90, June 2017.

[10] A. Singh, P. Vepakomma, O. Gupta, R. Raskar, “Detailed comparison of communication efficiency of split learning and federated learning,” ArXiv abs/1909.09145, 2019.

[11] P. Vepakomma, O. Gupta, T. Swedish, R. Raskar, “Split learning for health: Distributed deep learning without sharing raw patient data,” CoRR, vol. abs/1812.00564, 2018.

[12] Y. Duan and J. Wu, “Joint Optimization of DNN Partition and Scheduling for Mobile Cloud Computing,” in Proceedings of the 50th Int. Conf. on Parallel Processing (ICPP ’21), pp. 1–10, 2021.

[13] Z. Zhang, Q. Li, L. Lu, D. Guo and Y. Zhang, “Joint Optimization of the Partition and Scheduling of DNN Tasks in Computing and Network Convergence,” in IEEE Networking Letters, vol. 5, no. 2, pp. 130-134, 2023.