2021 IEEE International Conference on Smart Computing (SMARTCOMP) | 978-1-6654-1252-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/SMARTCOMP52413.2021.00024

Energy-Efficient Mapping for a Network of DNN Models at the Edge

Mehdi Ghasemi[1], Soroush Heidari[1], Young Geun Kim[2], Aaron Lamb[3], Carole-Jean Wu[1], and Sarma Vrudhula[1]

[1]Arizona State University, (mghasem1, sheidar1, carole-jean.wu, vrudhula)@asu.edu
[2]Soongsil University, younggeun.kim@ssu.ac.kr
[3]Qualcomm Inc., alamb@qti.qualcomm.com

***Abstract*—This paper describes a novel framework for executing a network of trained deep neural network (DNN) models on commercial-off-the-shelf devices that are deployed in an IoT environment. The scenario consists of two devices connected by a wireless network: a user-end device (U), which is a low-end, energy and performance-limited processor, and a cloudlet (C), which is a substantially higher performance and energy-unconstrained processor. The goal is to distribute the computation of the DNN models between U and C to minimize the energy consumption of U while taking into account the variability in the wireless channel delay and the performance overhead of executing models in parallel. The proposed framework was implemented using an NVIDIA Jetson Nano for U and a Dell workstation with Titan Xp GPU as C. Experiments demonstrate significant improvements both in terms of energy consumption of U and processing delay.**

***Index Terms*—edge computing, deep neural networks, energy**

I. Introduction

Deep Neural Networks (DNNs) are widely used for tasks, such as object detection, image classification, speech recognition, ranking and recommendation [11], [23], [12], [4]. To enable quality of experience, servers in the cloud are used to provide high performance, real-time processing. However, conventional cloud computing approaches come with large communication overhead. To overcome such overhead and to leverage the ever-increasing performance of devices at the edge [9], *edge computing* has been proposed where computation can occur on a local *cloudlet* (C) close to the first recipient of data (referred to as user-end device U) or directly on U itself [7], [25]. This style of execution is becoming increasingly common for applications, such as virtual/augmented reality, robots, and drones [24], [20], [3], [5]. The workloads of these applications now consist of a network of DNN models instead of a single model, as shown in Figure 1. For example, on a drone, objects on the scene are detected first and then different types of objects, such as cars or human, are classified. In such a workload, a first-stage DNN identifies objects and then activates other DNNs specialized for different types of objects.

Running an entire network of DNNs on U is often not feasible due to its limited energy capacity and functionality. Existing works addressed the energy constraint by distributing computation within a single DNN between a user-end device and the cloud [14], [17], [6] Considering the energy cost of computation and communication, the optimal partitioning leads to minimized energy consumption of U, which is crucial for the edge domain [3], [25]. However, the above techniques do not consider run-time variations, which have a large impact on the energy efficiency of U. In real use cases, many devices can be connected to and work with a cloudlet. Thus, the utilization of the cloudlet can significantly affect the execution time on C. In addition, the communication delay between devices especially over WiFi often exhibits stochastic behavior due to the distance between devices, signal interference, traffic in the network, etc. Even worse, the above techniques do not consider the overhead of parallel execution of DNNs. Due to limited computing resources on U, running two DNNs in parallel affects the performance of each other in the network of DNN models, resulting in higher energy consumption. For these reasons, the conventional partitioning approaches often fail to find the optimal mapping of flow graph nodes for participating devices, leading to energy inefficiency.

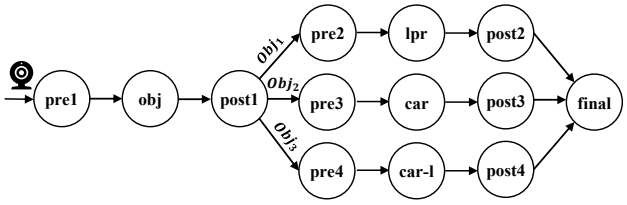

Fig. 1: An example of a network of DNN models. The first DNN detects objects and the parallel branches classify different types of objects identified in the first step.

This paper presents an energy-efficient solution for computation offloading targeting the energy minimization of U when executing a network of DNNs. At the beginning of processing each frame, the device U queries the status of C and the network delay. The run-time solution partitions the flow graph between U and C based on the response from C. Given a network of DNNs as a flow graph, the proposed approach constructs a modified weighted flow graph in such a way that the energy-optimal partitioning can be obtained using a min-cut procedure. The overhead of parallel execution of multiple DNNs is taken into account.

The proposed approach is demonstrated on a real platform, consisting of an NVIDIA Jetson Nano for U and a high performance workstation with an NVIDIA Titan Xp for C. Experiments demonstrate that our proposed design significantly

978-1-6654-1252-0/21/$31.00 ©2021 IEEE
DOI 10.1109/SMARTCOMP52413.2021.00024

improves the energy efficiency of U by an average of 42% and 64% over the baseline settings where all the computations are executed on U and C, respectively.

The contributions of this work are summarized as below:

- This paper provides a detailed characterization for executing a network of DNN models in a realistic edge environment. The results highlight the importance of considering run-time variations and parallel execution for optimal partitioning of the network of DNN models.
- This paper presents an energy-efficient partitioning solution between a user-end device and the cloudlet for running a network of DNN models based on the energy profiles of nodes in the flow graph.

II. Related Work

Offloading Algorithms: Kaya et al. [15] get the call graph of a Java program and bi-partition the nodes of the given graph using the FM algorithm [8]. The method was applied on conventional object recognition algorithms. Gao et al. [10] address the stochastic behavior of the execution times in applications, during code-offloading from a mobile device to a cloud server. The aforementioned stochastic behavior is modeled using a semi-Markov chain. Indeed, no formal optimization is involved in their approach.

Offloading for DNNs: Due to the computational-intensive nature of DNN models and the high power consumption of running these workloads on resource-constrained IoT devices, different computation offloading strategies have been proposed in the literature. The idea of collaborative execution between U and cloud server has been proposed first in [14] and then used in [17]. In fact, the DNN is partitioned between the cloud server and mobile device in the granularity of a singe layer. *JointDNN* [6] is a recently-proposed method for collaborative execution of DNNs. The presented method in [6] is based on the shortest path problem specifically for DNNs. In order to use the shortest path method to solve the partitioning problem, the number of nodes in the graph is duplicated. The generality of the shortest path solution has not been discussed. The studies [14] and [6], only considered a sequence of layers inside a single DNN. Consequently, those approaches would not be a solution to networks with parallel nodes.

MoDNN [18] and *DeepThings* [26] target the implementation of DNNs on resource-constrained devices such as mobile phones. These frameworks distribute the computation of DNNs/CNNs on a cluster of IoT devices connected using a wireless network based on partitioning the input data. *Autoscale* [16] presents a reinforcement learning solution to determine where to execute the whole DNN inference with the options of CPU and co-processors in the mobile device, or a locally connected device. It considers the resource interference and signal strength variation.

Prior works are limited to mapping a single DNN model. In this paper, we address a more complex and challenging problem of mapping a *network* of DNN models across edge devices. The proposed solution considers, among other things, the overhead of parallel execution of computationally-intensive nodes and the effect of variations of the communication delay and the computation time of the cloudlet. These aspects have not been included in earlier works.

III. Models and Problem Formulation

This section first describes the hardware components and the workload model. Next, the delay and energy models are explained followed by a formal definition of the problem.

A. Hardware: The system consists of three components: a user-end device U with limited energy capacity and limited performance, a cloudlet C that can deliver much higher performance than U and is not energy-constrained, and a wireless medium W connecting these two devices.

B. Workload Model: The workload consists of DNN models for various functions, such as object detection, forming a network of DNN models. The workload is modeled as a flow graph $G = \{V_G, E_G\}$, where the vertices V_G represent the computational blocks of different models, and the edges E_G represent data dependencies.

C. Delay Model: The total completion time of processing the flow graph for an input data is dependent on computation and communication delays. The computation delay of node i in the flow graph on U and C are denoted with $\delta(i|U)$ and $\delta(i|C)$. The term $\delta(i|C)$ depends on the utilization (i.e., the number of concurrent requests) of the cloudlet. These values are obtained during the profiling step. The wireless communication delay between devices exhibits significant variations and is expressed as (see [2])

$$\delta^i_{U,C} = S_i/B + RTT, \tag{1}$$

where RTT is the roundtrip time (ms), B is the bandwidth (MBps), and S_i is the size of the data in node i (KB).

D. Energy Model: The energy consumption is the product of delay and power dissipation. The goal of this work is to determine the optimal assignment of computation nodes (V_G) to the devices that minimizes the energy consumption of U when executing a given workload. Given the flow graph $G = (V_G, E_G)$, let $a_i = 0$ or 1 if $i \in V_G$ is to be executed on U or C, respectively. The total energy consumption of U consists of the computation and communication energy. $E_U(i|U)$ denotes the computation energy consumption of U when the computation block represented by node $i \in V_G$ is executed on U. Therefore, $E_U(i|U)$ is expressed as:

$$E_U(i|U) = \delta(i|U) \times P_{comp}(i), \tag{2}$$

where $\delta(i|U)$ is the delay of running node i on U, $P_{comp}(i)$ is the average power consumption of computation on U for node i. Similarly, $E_U(i|C)$ denotes the energy consumption of U when block i is executed on C and is expressed as:

$$E_U(i|C) = \delta(i|C) \times P_{idle}. \tag{3}$$

Thus, it includes the energy consumption when U is idle and the computation is performed on C.

Based on the assignment variable a_i, the computation energy of node i denoted by $E_U(i)$ can be written as follows:

$$E_U(i) = a_i E_U(i|C) + (1 - a_i) E_U(i|U). \tag{4}$$

$E_U(i|d_s, j|d_d)$ denotes the energy consumption due to data communication associated with edge $(i,j) \in E_G$, when node i is executed on device d_s and node j is executed on device d_d, for $d_s, d_d \in (U, C)$.

There are four possibilities in the assignment of nodes to devices for an edge of the graph. The terms $E_U(i|U, j|U)$ and $E_U(i|C, j|C)$ are equal to zero since nodes i and j are both executed on U or C and there is no internal energy overhead in executing both i and j on U.

The term $E_U(i|U, j|C)$ is the energy consumption of U when node i is executed on U and node j is executed on C. In the flow graph, the result of node i is the input to node j. Thus, the term $E_U(i|U, j|C)$ is the energy consumed by U when transferring the results of node i from U to C.

The value of $\delta_{U,C}$ affects the total energy consumption of U where there is a need for communication and transfer of results between devices. The two cases for the communication are $E_U(i|U, j|C)$, $E_U(i|C, j|U)$. For instance, the term $E_U(i|U, j|C)$ can be defined as below:

$$E_U(i|U, j|C) = (S_i/B + RTT) \times P_{comm}, \tag{5}$$

where P_{comm} is the average power consumption of U during data transfer.

The communication energy consumption of U can now be expressed in terms of the above quantities and the decision variables, as follows:

$$E_U(i,j) = a_i(1-a_j)E_U(i|C, j|U) + (1-a_i)a_j E_U(i|U, j|C). \tag{6}$$

E. Problem Formulation: As stated earlier, the objective here is to determine the best assignment of computation nodes to the devices (finding a_i for node i in V_G) that minimizes the energy consumption of U while executing the workload. The optimization of total energy consumption during the workload execution including the computation energy and communication energy can be expressed as below:

$$Minimize \quad (\sum_{i \in V_G} E_U(i) + \sum_{(i,j) \in E_G} E_U(i,j)) \tag{7}$$

IV. Motivation for proposed approach

The proposed approach to minimize the energy consumption of U takes into account several important factors: (1) the execution time and the corresponding power consumption of each node in the flow graph on U and C; (2) the amount of data that needs to be transferred between U and C which depends on the flow graph node; (3) the variations in the network delay, and (4) the overhead when executing several graph nodes in parallel.

Factors (1) and (2) are accounted for through extensive profiling of the computation nodes on the devices. Factor (3) is considered by sampling the round-trip time (RTT) at the start of determining the optimal partition, and by using Equation (1) to estimate the communication delay. The importance of (4) is demonstrated in Table I and Table II. Two pairs of DNN models (lpr, car) and (car, car-l) (shown in Figure 1) were executed sequentially and in parallel on U (NVIDIA Jetson Nano). Lpr is a relatively light-weight DNN comprising of LPRNet [27], and car and car-l are both heavy-weight models with ResNet structure [1].

Tables I and II show the computation delay of each model under the two different scenarios ($\delta_{sol}(i|U)$ and $\delta_{par}(i|U)$). The key observation is that in both cases, ignoring the overhead due to parallel execution severely underestimates the computation time and therefore energy consumption. The consequence would be a highly sub-optimal partitioning solution. Our approach to account for (4) is described in the following section.

DNN model	$\delta_{sol}(i\|U)$ (solo)	$\delta_{par}(i\|U)$ (parallel)
lpr	18.5 ms	43.1 ms
car	97.5 ms	107.7 ms

TABLE I: The parallel execution of a lightweight model (lpr) and a complex one (car) on U affects each other. However, it does not result in significant increase in the critical path.

DNN model	$\delta_{sol}(i\|U)$ (solo)	$\delta_{par}(i\|U)$ (parallel)
car	97.5 ms	172.2 ms
car-l	90.6 ms	183.2 ms

TABLE II: The parallel execution of two complex DNN models on U affects the performance of both models and the critical path significantly.

V. Proposed Approach

To minimize the energy consumption of the user-end device (U), a run-time decision is made at the start of processing each frame. For each frame, U queries the status of C (execution time for each node of the flow graph on C and network delay). Based on the data received, and the known energy costs of running nodes on U itself, U decides how to partition the computation.

Given the energy profiles, the problem can be cast as a graph bi-partitioning problem that can be solved optimally using the *maxflow* algorithm. This was demonstrated in [22], which addressed a different problem of hardware-software bi-partitioning for reconfigurable systems. In that work, the computation blocks of a workload, represented as a control-dataflow graph (CDFG), are statically assigned and executed on a microprocessor or on a dynamically reconfigurable FPGA, with the objective of minimizing the total energy consumption. Included in their formulation are the computation and communication time, as well as the energy consumption of computation blocks on both devices.

The method presented in [22] involves transforming a given computation graph $G = (V_G, E_G)$ to another graph $\tilde{G}(\tilde{V_G}, \tilde{E_G})$, such that the minimum cut in $\tilde{G}$ corresponds to the optimal bi-partitioning of G. The proof of optimality of this polynomial-time transformation appears in [21]. Here only the basic steps involved in the transformation are presented in the context of computation offloading problem [22].

The transformation involves (1) adding two distinguished nodes, s (source) and t (sink) to the set of nodes, (2) adding

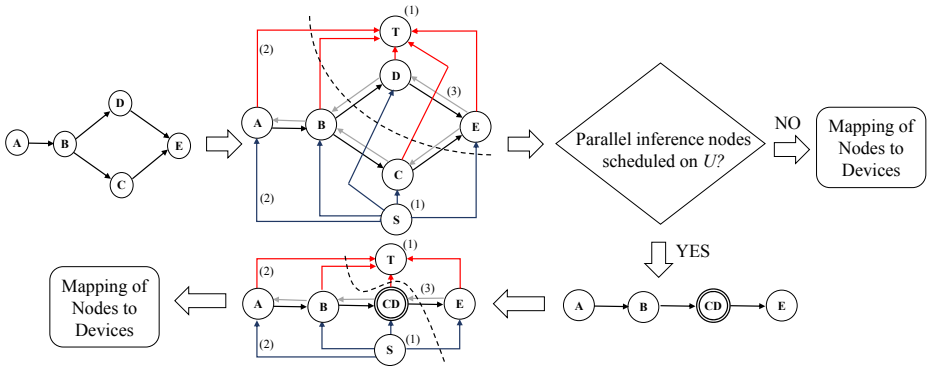

Fig. 2: The flow of the partitioning algorithm. First, the proposed algorithm finds the min-cut in the transformed flow graph based on solo energy profiles of the graph nodes. If parallel inference nodes are scheduled on U, another checking is done based on the energy profile of super nodes.

an edge from s to each node and from each node to t; (3) replacing each edge in G by a pair of edges. Figure 2 shows an example of this transformation with the steps labeled. First, the value of $E_U(i|C)$ is used to assign the cut values of links between source and all nodes in the original graph. Likewise, $E_U(i|U)$ is used to obtain the cut value between the sink and the other nodes. The values of capacities $\tilde{C}_E$ from the source to all nodes and from all nodes to the sink are assigned to the edges as follows:

$$\begin{aligned}\tilde{C}_E(s,i) &= E_U(i|C) + \sum_{(j,i)\in E_G} E_U(j|C,i|C),\\ \tilde{C}_E(i,t) &= E_U(i|U) + \sum_{(j,i)\in E_G} E_U(j|U,i|U).\end{aligned} \tag{8}$$

Let

$$\begin{aligned}\tilde{C}^F_E(i,j) &= E_U(i|U,j|C) - E_U(i|C,j|C), (i,j)\in E_G,\\ \tilde{C}^B_E(i,j) &= E_U(j|C,i|U) - E_U(j|U,i|U), (j,i)\in E_G.\end{aligned} \tag{9}$$

Then, the capacity of edge $\tilde{C}_E(i,j)$ can be obtained as below:

$$\tilde{C}_E(i,j) = \tilde{C}^F_E(i,j) + \tilde{C}^B_E(i,j). \tag{10}$$

The relations $E_U(i|U,j|C) \geq E_U(i|C,j|C)$ and also $E_U(j|C,i|U) \geq E_U(j|U,i|U)$ hold in the computation offloading problem. Thus, the cut values will not be negative. Finally, the algorithm returns the new set of nodes and edges and their cut values which is used in the partitioning algorithm.

The partitioning algorithm applies the transformations described above to the flow graph $G = (V, E)$. Then, a minimum capacity cut on the transformed graph $\tilde{G}$ is obtained by the maxflow algorithm. The assignment variable a_i is found for all the nodes based on the cut, which divides the nodes into two sets. Note that the complexity of maxflow algorithm is $O(|{V_G}^3|)$ [19].

If two parallel nodes end up being scheduled on U, the solution should be invalidated. In such a situation, the effect of parallel nodes is taken into account by forming a super node, as shown in the lower part of Figure 2. Using the energy profiles of super nodes, the partitioning algorithm generates a new partitioning in which the nodes inside a super node are scheduled on the same device. In the general approach, every combination of more than two nodes running in parallel on a same device should be considered. Since the maximum level of parallelism is limited to four threads in most commodity devices, the overhead of decision making is not significant. The implementation of parallel nodes was done using input and output queues for each DNN. When there is a request inside the input queue, the pre-processing function (pre1 in Figure 1) is executed. When post-processing is completed for the request, the output queue is written, which then triggers the input queue of the parallel branches in the next stage.

The approach based on the maxflow technique can be used for any general graph structure. The cost function in the optimization problem should be additive in the general case of graph structure. This is true for the case of energy consumption. In the case of delay optimization, the maxflow method gives the optimal solution for the line graph.

VI. Experimental Methodology

Device Setup: An NVIDIA Jetson Nano was selected as the user-end device (U) and a Dell workstation with an NVIDIA Titan Xp as the cloudlet device (C). The devices were connected using a Synology RT2600ac wireless access point. The total power consumption of U was measured using a Monsoon Power Monitor.

Workload: The workload consists of several DNN models as shown in Figure 1. For each detection model, there are pre-processing and post-processing steps which are done before and after the inference step. First, the objects are detected using Tiny YOLO v3 model [23]. The result of pre-processing is fed into other branches for further processing. This emulates tasks, such as, recognition of car license plates and models [27], [1]. The final processing is performed based on the results of all the models.

Performance Profiling: As expected, the execution time of the inference nodes was the least on C. The utilization of CPU cores and GPU on U was checked while running the workload. The inference nodes fully utilized the GPU cores for the obj, car, and car-l nodes (achieving 99% utilization of the GPU) where lpr utilized 77% of the GPU. The pre-processing and post-processing nodes utilized one of the CPU cores.

Power Profiling: The highest power consumption ($P_{comp}(i)$) is related to running the inference nodes since they utilize the

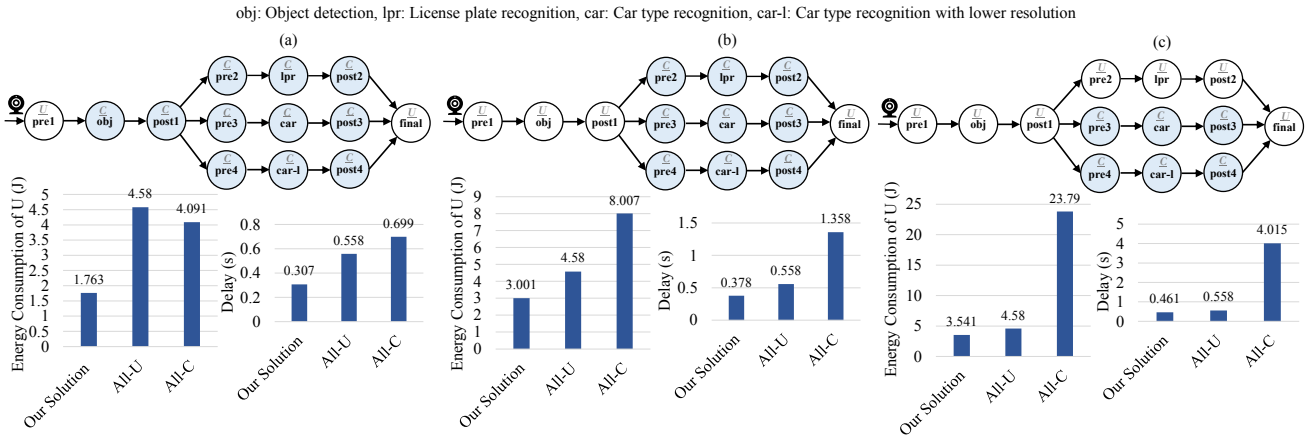

Fig. 3: Partitioning solutions in different scenarios of bandwidth ((a): 12 MBps, (b): 6MBps, (c): 2MBps). The white and blue nodes refer to running on U and C respectively. The energy consumption of U and delay are compared with All-U and All-C cases in different scenarios.

GPU. The average communication power consumption was measured as 5.94W which makes offloading more beneficial when compared with the inference power consumption of 8.58 W on U.

Network Delay Profiling: The size of input data decreases throughout the execution of the flow graph. The ideal communication bandwidth and the round-trip time between the devices were measured as 12 MBps and 4.5ms respectively. In the presence of obstacles or larger distance between the devices, the observed bandwidth and round-trip time degraded to 1 MBps and 25 ms respectively.

VII. Experimental Results

Figure 3 (a) shows the proposed partitioning solution across two devices, assuming the ideal bandwidth and execution time on C ($B = 12MBps$). In this case, the first pre-processing node (pre1) was scheduled on U while all other nodes were executed on C. The reason for executing pre1 on U is that the overhead of sending the input data is higher than running the node on C, since U needs to send 8100 KB of input data for node pre1. Figure 3 (a) shows that the optimal partition obtained by our maxflow technique improves both the energy consumption of U and the overall performance when compared to the default solutions: All-U and All-C. In particular, under the the ideal conditions of execution time and bandwidth, *the proposed solution improves the energy of U by 61% and delay by 45% when compared to the All-U solution. When compared to the All-C solution, the proposed approach improves the energy and delay by 57% and 56%, respectively*.

Figure 3 (b) shows the partitioning result for the reduced bandwidth case. When the bandwidth was 6 MBps, the nodes obj and post1 were also scheduled on U since the overhead of sending the output of pre1 was higher than running the obj and post1 on U. However, the remainder of the parallel nodes were scheduled on C since the input size of these parallel models was smaller for parallel branches. In this case, *the proposed solution improves the energy of U by 34% and delay by 32% when compared with the All-U solution. In comparison with the All-C solution, the proposed approach improves the energy and delay by 62% and 72%, respectively*.

Figure 3 (c) shows the results when the bandwidth is 2 MBps. In this case, the first three nodes were mapped on U. Moreover, the nodes pre2, lpr, and post2 were also scheduled on U. This is due to the fact that the execution time of running lpr on U is lower, compared with the overhead of data transfer. It is, however, beneficial to send the data and offload the computation of the other two parallel branches to the cloudlet (C) since it takes more time to finish the computation on U.

Figure 4 shows the result of partitioning when the bandwidth was 12 MBps and there were concurrent requests for running the models on C. The difference in the partitioning solution is in the nodes lpr and post2 where these nodes were scheduled on U when the delay of running lpr on C increased. However, the inference nodes car and car-l were still scheduled on C since the execution time of running these nodes on U is significantly higher compared with running them on C. In this case, *the proposed solution improves the energy consumption of U by 49% and delay by 25% in comparison with the All-U solution. Compared with the All-C solution, our solution leads to 51% and 50% improvement in terms of energy consumption and processing delay.*

Comparison with Line Graph Approach: Figure 5 shows the comparison of our solution with the previous work on the collaborative execution of DNNs between the user-end device and the cloudlet. A partitioning algorithm in the granularity of a single layer inside a DNN (line flow graph) has been presented in [6] and therefore this approach would not be a solution for flow graphs with parallel branches. However, for the fair comparison, the general flow graph was converted to a line graph and the previous methodology was applied on the obtained line graph. In fact, the parallel nodes in each level of the flow graph are combined and considered as a single

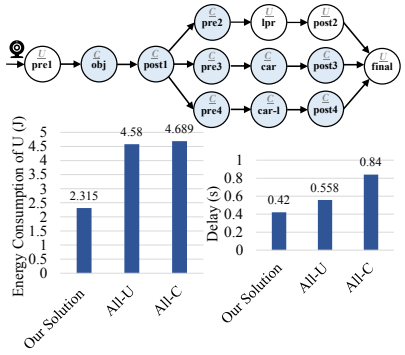

Fig. 4: Partitioning result in the case of running concurrent models on C and bandwidth of 12 MBps.

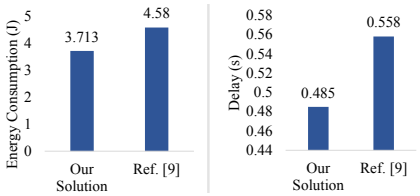

Fig. 5: Comparison of our solution with [6] in terms of energy consumption and delay in the case of bandwidth $B = 2MBps$ and concurrent requests on the cloudlet.

node forming a line graph. Using this approach, the nodes on the parallel branches need to be scheduled on one device. However, as the result of our proposed solution, the nodes in the first branch (lpr) were scheduled to be executed on U (due to higher cost of execution on C) where car and car-l were executed on C. Therefore, our solution leads to 19% and 13% improvement in terms of energy consumption and delay.

The Effect of Parallel Execution: The effect of parallel execution can be observed in the All-U schedule. In this schedule, the prediction of naive graph partitioning for energy consumption based on solo profiles was 3.702 J. However, the actual energy consumption of All-U schedule was 4.58 J. Therefore, there was about 19% difference between the actual energy consumption of running all nodes on U and that of naive partitioning based on solo profiles. This result shows the importance of considering the parallel execution in the partitioning solution.

Overhead of Decision Making: The overhead of computing the partition for every frame was measured on U. For the graph with 13 nodes, it took 4.7 ms to find the assignment of nodes to devices at run-time. The average power consumption to compute the optimal partition was 0.93 w. Therefore, each decision making at the beginning of each frame consumed 4.37 mJ. The energy overhead of decision making is negligible (less than 1%) compared with the energy consumption of frame processing in the optimal setting. The decision making algorithm was implemented in Python based on finding the min-cut in the input flow graph using *networkx* package [13].

VIII. Conclusion

This paper presented a novel approach to execute a network of DNN models on commercial-off-the-shelf devices. The goal is to partition the network of DNNs between the user-end device and the cloudlet to minimize the energy consumption of the user-end device. The proposed partitioning algorithm considers the computation cost on devices, communication cost, and the overhead of executing DNNs in parallel. Experiments on commodity platforms demonstrate 42% and 64% improvement in terms of energy consumption compared with running the entire workload on U or C, respectively.

Acknowledgement

This research was supported in part by NSF Grant #2008244, and by the Center for Embedded Systems, NSF Grant #1361926.

References

[1] Car recognition. https://github.com/foamliu/Car-Recognition, 2021.
[2] F. Adelstein et al. *Fundamentals of Mobile and Pervasive Computing*, volume 1. McGraw-Hill New York, 2005.
[3] K. Apicharttrisorn et al. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *SenSys*, 2019.
[4] E. Bank-Tavakoli et al. Polar: A pipelined/overlapped FPGA-Based LSTM accelerator. *TVLSI*, 2019.
[5] R. Dedinsky et al. A dependable detection mechanism for intersection management of connected autonomous vehicles. In *ASD*, 2019.
[6] A. E. Eshratifar et al. JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services. *TMC*, 2019.
[7] M. Farhadi et al. Enabling incremental knowledge transfer for object detection at the edge. In *CVPRW*, pages 396–397, 2020.
[8] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC*, 1982.
[9] C. Gao et al. A study of mobile device utilization. In *ISPASS*, 2015.
[10] W. Gao et al. On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *ICNP*. IEEE, 2014.
[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
[12] U. Gupta et al. The architectural implications of Facebook's dnn-based personalized recommendation. In *HPCA*, 2020.
[13] A. Hagberg et al. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab., 2008.
[14] Y. Kang et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *ASPLOS*, pages 615–629. ACM, 2017.
[15] M. o. Kaya. An adaptive mobile cloud computing framework using a call graph based model. *Network and Computer Applications*, 65, 2016.
[16] Y. G. Kim and C.-J. Wu. Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning. In *MICRO*, 2020.
[17] E. Li et al. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *MECOMM*, pages 31–36, 2018.
[18] J. Mao et al. Modnn: Local distributed mobile computing system for deep neural network. In *DATE*, pages 1396–1401. IEEE, 2017.
[19] B. M. Moret and H. D. Shapiro. Algorithms from p to np. Technical report, Benjamin-Cummings Publishing Co, 1991.
[20] M. Radovic, O. Adarkwa, and Q. Wang. Object recognition in aerial images using convolutional neural networks. *Journal of Imaging*, 2017.
[21] D. Rakhmatov. *Modeling and Optimization of Energy Supply and Demand for Portable Reconfigurable Electronic Systems*. PhD thesis, University of Arizona, 2002.
[22] D. N. Rakhmatov and S. B. Vrudhula. Hardware-software bipartitioning for dynamically reconfigurable systems. In *CODES+ISSS*, 2002.
[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, pages 779–788, 2016.
[24] A. Uçar et al. Moving towards in object recognition with deep learning for autonomous driving applications. In *INISTA*, 2016.
[25] C.-J. Wu et al. Machine learning at Facebook: Understanding inference at the edge. In *HPCA*, 2019.
[26] Z. Zhao et al. Deepthings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters. *TCAD*, 2018.
[27] S. Zherzdev and A. Gruzdev. Lprnet: License plate recognition via deep neural networks. *arXiv preprint arXiv:1806.10447*, 2018.