IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 12 January 2023, accepted 5 February 2023, date of publication 13 February 2023, date of current version 1 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3244497

RESEARCH ARTICLE

DNN Partitioning for Inference Throughput Acceleration at the Edge

THOMAS FELTIN[1], LÉO MARCHÓ[2], JUAN-ANTONIO CORDERO-FUERTES[1], (Member, IEEE), FRANK BROCKNERS[2], AND THOMAS H. CLAUSEN[1], (Senior Member, IEEE)

[1]École Polytechnique, 91128 Palaiseau, France
[2]Cisco Systems, San Jose, CA 95134, USA

Corresponding author: Thomas Feltin (thomas.feltin@polytechnique.edu)

ABSTRACT Deep neural network (DNN) inference on streaming data requires computing resources to satisfy inference throughput requirements. However, latency and privacy sensitive deep learning applications cannot afford to offload computation to remote clouds because of the implied transmission cost and lack of trust in third-party cloud providers. Among solutions to increase performance while keeping computation on a constrained environment, hardware acceleration can be onerous, and model optimization requires extensive design efforts while hindering accuracy. DNN partitioning is a third complementary approach, and consists of distributing the inference workload over several available edge devices, taking into account the edge network properties and the DNN structure, with the objective of maximizing the inference throughput (number of inferences per second). This paper introduces a method to predict inference and transmission latencies for multi-threaded distributed DNN deployments, and defines an optimization process to maximize the inference throughput. A branch and bound solver is then presented and analyzed to quantify the achieved performance and complexity. This analysis has led to the definition of the acceleration region, which describes deterministic conditions on the DNN and network properties under which DNN partitioning is beneficial. Finally, experimental results confirm the simulations and show inference throughput improvements in sample edge deployments.

INDEX TERMS Distributed artificial intelligence, edge computing, scheduling and task partitioning.

I. INTRODUCTION

Connected devices generated an estimated 2.5 quintillion bytes of data every day in 2020.[1] In this context, Artificial Intelligence (AI) is one of the technologies that can best cope with the always growing amount of produced data, in a range of fields such as object detection in computer vision, facial recognition, speech recognition, natural language processing, or autonomous driving. AI, and specifically Deep Learning (DL), requires, in addition to large amounts of data, significant capabilities in processing power and memory, which makes cloud computing the *de facto* hosting solution for AI workloads. Consequently, in 2019, 96% of AI tasks were run in the cloud.[2]

AI applications can have strong latency constraints, *e.g.*, in autonomous driving, manufacturing monitoring, or any real-time inference involving a real world interaction. For example, average response times in autonomous driving are required to be under 100 milliseconds [1], making round trip times to the cloud too long for such applications. Similarly, some applications may introduce a significant link usage on the network, *e.g.*, in the case of high quality video processing in computer vision, which prevents them from running in the cloud. Data privacy policies or data protection regulations may also prohibit data from leaving specific environments. This encourages AI deployments in edge computing, which

The associate editor coordinating the review of this manuscript and approving it for publication was Rodrigo S. Couto.

[1]According to the *Data Never Sleeps* annual study from Domo https://www.domo.com/learn/infographic/data-never-sleeps-8

[2]According to a survey from Nucleus Research Inc., 96% of Deep Learning based applications ran in the cloud in 2019. https://d1.awsstatic.com/whitepapers/Deep%20learning%20on%20AWS.pdf

consists of relocating computation tasks from data centers closer to edge devices, *i.e.*, in proximity to the data sources.

In comparison with public clouds, the edge is a resource-constrained environment with important limitations [2], and developing AI for the edge is therefore intrinsically different from developing AI for the cloud. Analyzing data close to its source can be challenging because of lower device capacities, inelasticity of the available resources, and heterogeneity of edge networks. Lowering the computing capacity implies lowering the achievable inference throughput, further complicating relocation of heavy workloads from the cloud to the edge. In some applications, the inference throughput, *i.e.*, the achievable inferences per second of a DNN, can be linked to the DNN accuracy, *e.g.*, with object tracking in video streams, where dropping the inference rate from 15 to 5 inferences per seconds has shown to lower the F1 score[3] of an object detector by 10% [3]. In other words, deployments with low inference throughput can cause critical information loss, *e.g.*, loss of detections in video surveillance applications.

There are two main available methods to meet performance requirements on constrained edge networks: model compression or hardware acceleration. *Model compression* consists of reducing and optimizing neural networks to a light-weight, often under-performing, version of the model. Standard model compression methods include pruning [4], *i.e.*, removing unnecessary parameters in the model, quantization [5], *i.e.*, reducing the allocated memory to store the model, and knowledge distillation [6], *i.e.*, training a smaller neural network with knowledge extracted from a larger one. Model optimization requires extensive design efforts and can be an impediment to model accuracy. *Hardware acceleration* on edge devices implies finding hardware which can be both power efficient and light-weight, while still being able to run inference with sufficient performance. There are several types of candidate hardware for acceleration at the edge, varying in efficiency and specificity, but standard processing units fail to meet expectations and dedicated hardware such as ASICs have been found to be unpractical and expensive [7].

DNN partitioning is a complementary method for accelerating inference by leveraging the multiplicity of existing devices on edge networks to distribute the inference computation – which can be used alone, or in conjunction with the two other methods. DNN partitioning consists of considering a neural network as a pipeline to segment into partitions, and distributing these partitions on edge devices. The placement of these partitions is based on both the DNN and the underlying network characteristics. DNN partitioning relies on the identification of split points, which are points in the model graph where the model is separated into partitions. During run-time, partitions are run sequentially, each sending intermediary inference results to the next partition. This allows each partition to start computing the next input data while the other devices continue processing the offloaded one, hereby improving the inference throughput, as shown in figure 1. In the remainder of this paper, DNN partitioning is illustrated through the example of real-time inference on video streams.

[3]The F1 score is a measure of a model's accuracy on classification tasks.

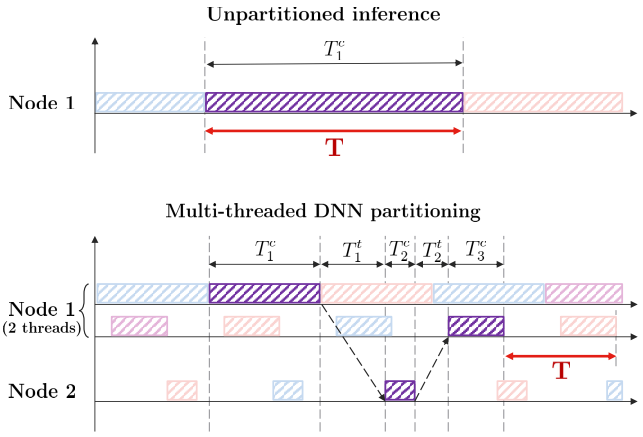


FIGURE 1. Timeline of multi-threaded inference partitioning over 2 devices compared with unpartitioned inference. The figure shows the computation (T_i^c) and transmission (T_i^t) latencies. The inference throughput is improved by partitioning, and bound by the slowest element in the network (T).

A. RELATED WORK

Methods for DNN partitioning, derived from mobile edge-cloud offloading, seek to optimize varying performance indicators, such as computing latency, energy consumption, resource utilization, cost, or throughput. DNN partitioning relies on the association of one or several of these metrics to define an optimization goal, and a method which exploits this metric to find partitioning schemes. For example, Neurosurgeon [8] seeks a single split point, keeping the first partition at the edge and offloading the second partition to the cloud, to minimize latency and energy consumption. Applications in IoT have also considered the joint partitioning and offloading of several DNNs to optimize energy, delay, and/or cost, using different solvers such as SPSO-GA [9] combining Particle Swarm Optimization (PSO) and Genetics Algorithm (GA), or DDPQN [10] which uses Deep Reinforcement Learning to find partitioning schemes. Other examples include DINA [11], which defines an Integer Non Linear Programming (INLP) problem, and uses a matching theory based solver, to optimize delay and resource utilization in fog networks. Methods optimizing inference throughput include DNN surgery [12], which uses the lower size of intermediary DNN layer outputs to partition the inference computation between the edge and the cloud. Both algorithms create two partitions, one at the edge and the other in the cloud. Other studies in the field of mobile edge-cloud [13] also include multi-threaded computation in the cloud to further accelerate the overall inference throughput. Relying on the cloud for the second partition inference computation assumes good network connectivity, which can be uncertain with the poor connectivity of some isolated edge deployments. Edgent [14] adapts the previous methodology to mobile devices and

available edge servers, relaxing the constraint of offloading to the cloud. This method is linked to a specific training and model architecture, and still limits the partitioning to two partitions. Other methods have considered multiple split points, *e.g.*, IONN [15], which describes the problem of partitioning to minimize latency and energy consumption, also taking into account the time to upload the models to edge servers.

These methods are either interacting with remote clouds, or are limited in the nature of the partitioning, *e.g.*, required to split in exactly two parts, with or without multi-threading. None of the listed DNN partitioning studies considers both multiple split points and multi-threading.

B. CONTRIBUTIONS

This paper presents a distributed inference framework to maximize the inference throughput of real-time DNN computation on streaming data, with multiple split points and multi-threaded partitioning. The contributions of this paper are the following:

- A model for computing and transmission latencies of a distributed DNN, through which the expected inference throughput of a given partitioning scheme can be estimated.
- Formulation of the optimization problem for DNN partitioning, implementation of a branch and bound solver for this problem, and evaluation of its complexity.
- Simulations to explore the possible performance improvements with varying network and DNN properties.
- The identification of deterministic regions in the network and DNN properties leading to the existence of optimal partitionings and the cost to compute such solutions, *i.e.*, the conditions under which DNN partitioning is beneficial.
- Experimental results illustrating the regions defined in the simulations, as well as the prediction accuracy and final inference throughput acceleration for homogeneous and heterogeneous environments.

For the remainder of this paper, to avoid confusion, the partitioned deep neural network will be referenced as *DNN*, and the term *network* will denote the underlying physical communications network.

C. PAPER OUTLINE

The remainder of this paper is organized as follows. Section II presents background of Deep Learning and how DNNs are represented in this study. Section III details the inference and transmission latency prediction methods required for section IV which defines the optimization problem, as well as the branch and bound algorithm to take partitioning decisions. Section V presents simulation results to quantify the performance and complexity of the branch and bound algorithm, as well as conditions leading to the existence of a partitioning which improves the inference throughput in a homogeneous network. Section VI presents experimental results confirming the simulations of section V, and shows inference throughput improvements on a heterogeneous experimental set-up. Finally, section VII discusses the results and expands on the broader applicability of this method.

II. BACKGROUND: DEEP LEARNING

Within the field of Machine Learning, Deep Learning (DL) refers to methods relying on the use of DNNs, *i.e.*, artificial neural networks (or related machine learning methods) containing more than one hidden layer. One of the earliest references to deep learning architectures was published in 1967 [16], but it was not until the 2010s that DL gained traction, first in speech recognition, then in object detection in images, for the ability to capture complex relationships in high dimensional in data. Deep Learning is estimated to represent the majority of AI applications in 2022, with more than 75% of organizations using DNNs for applications that could use classical methods.[4] DL methods have been defined as "*techniques for machine learning in which hypotheses take the form of complex algebraic circuits with tunable connection strengths*" [17]. These algebraic circuits are usually organized into layers, which form steps in the computation. The term *deep* refers to algorithms consisting of more than one layer.

There are two main DNN structures: feed-forward neural networks and recurrent neural networks. Both types of DNNs can be represented as computation graphs, with the main difference being that feed-forward networks can be represented as direct acyclic graphs (DAGs), while recurrent neural networks may contain cycles. Recurrent neural networks are built for sequences of data, where each data point has a potential dependency with previous data points in the sequence, while feed-forward neural networks do not consider interactions between points in a data sequence.

The remainder of this paper will consider the case of feed-forward DNNs.[5]

In the DAG representing the computation of a DNN, each node represents a layer, *i.e.*, the elementary operation on the input. There are several types of layers depending on the applied operation, *e.g.*, fully-connected, convolutional, pooling, batch normalization, etc. Each layer, *i.e.*, each node in the graph, applies a function to its inputs followed by a non linear operation, called an *activation function*. An example computation graph structure is shown in figure 2. Developing and training a DNN consists in defining a DAG, and adjusting the weights of these operations to fit the input data to a desired output inference.

III. DISTRIBUTED INFERENCE MODELING

This section presents a model to represent DNN and network properties in order to predict computation latencies,

[4]https://gartner.com/smarterwithgartner/gartner-predicts-the-future-of-ai- technologies

[5]Feed-forward DNNs are well suited for inference on video streaming data.

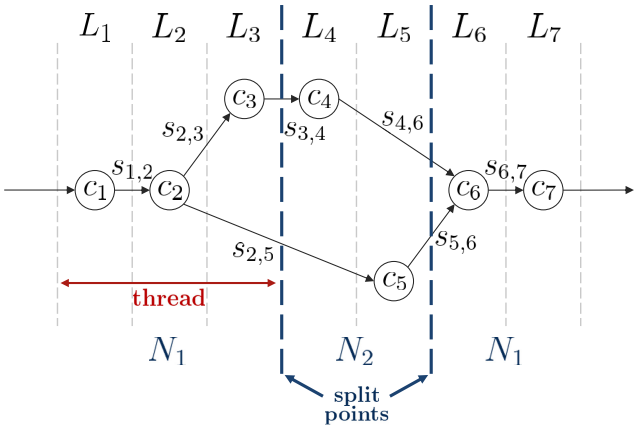


FIGURE 2. Example computation graph of a feed-forward neural network. Each layer L_i runs c_i operations, sends $s_{i,j}$ amount of data to the following layer, and is placed on a node N_p (see section III-A).

TABLE 1. Summary of mathematical notations.

Symbol	Description
$\mathcal{G}$	The communications network graph
$\mathcal{G}_A$	The feed-forward network graph
$\mathcal{L}$	The set of layers in the DNN graph
$\mathcal{E}$	The set of connections between layers in the DNN graph
$\mathcal{N}$	The set of compute nodes
$\mathcal{V}$	The set of links between compute nodes
L	The number of layers in the DNN
N	The number of compute nodes in the network
L_i	The i-th DNN layer
N_a	The a-th compute node
c_i	The compute consumption of layer L_i in FLOPs
$s_{i,j}$	The size of the data transiting between layers L_i and L_j
η_a	The processing rate of node N_a
$\theta_{a,b}$	The link bandwidth between nodes N_a and N_b
$\mathbf{P}$	The partitioning matrix
$\mathbf{H}$	The inverse processing rate matrix
$\mathbf{c}$	The layer consumption vector
$\mathbf{S}$	The transmission size matrix
$\mathbf{\Theta}$	The inverse link bandwidth matrix
$T^c(N_a)$	The inference latency on node N_a
$\mathbf{T}^c$	The inference latency matrix
$T^t(N_a, N_b)$	The transmission latency on link (N_a, N_b)
$\mathbf{T}^t$	The transmission latency matrix
C	The inference throughput
p_s	The partial placement list of size s
$\tilde{p}_s$	The padded version of placement list p_s
S	The maximum allowed number of split points
α	The processing rate to link throughput ratio

transmission latencies, and the final inference throughput of a given partitioning solution.

A. DNN AND NETWORK REPRESENTATION

A feed-forward DNN of N layers is modeled as a DAG $\mathcal{G}_A = (\mathcal{L}, \mathcal{E})$ with $\mathcal{L} = (L_1, \ldots, L_L)$ the layers of the DNN. Edges $(L_i, L_j) \in \mathcal{E}$ are the connections between layers L_i and L_j. Each layer L_i has an associated compute consumption c_i, measured in the number of floating-point operations required to compute a forward pass through the layer. Edges (L_i, L_j) are assigned a weight $s_{i,j}$ corresponding to the size of the data transiting between layers L_i and L_j in bytes.

The physical network is modeled as a fully connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{V})$ where $\mathcal{N} = (N_1, \ldots, N_N)$ is the set of compute nodes and $\mathcal{V}$ is the set of links between nodes. It is assumed that compute nodes $N_1, \ldots, N_N$ have processing rates $\eta_1, \ldots, \eta_N$, respectively, measured in floating-point operations per second. Finally, the link throughput between two adjacent nodes N_a and N_b is denoted as $\theta_{a,b}$, measured in bytes per second. Every node N_i is connected to itself with infinite throughput to represent the loopback link, and links are assumed to be symmetrical, *i.e.*, $\theta_{a,b} = \theta_{b,a}$.

Partitionings are defined as maps $P : \mathcal{L} \to N$, *i.e.*, a partitioning assigns a node number to each layer in the DNN. A partitioning can be described as a matrix $\mathbf{P}$ of dimension $(N \times L)$, N being the number of nodes on the network and L the number of DNN layers and, with $\mathbf{P}_{a,i} = 1$ if layer L_i is placed on node N_a, 0 otherwise. With the example given in figure 2, $L = 7$ layers and $N = 2$ compute nodes, the displayed partitioning with layers $\{L_1, L_2, L_3, L_6, L_7\}$ on node N_1, and layers $\{L_4, L_5\}$ on node N_2, is represented as:

$$\mathbf{P} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Given a partitioning, a *thread* is defined as a group of consecutive layers between two split points, run sequentially on the same node. In the example above, node N_1 will run two threads, the first containing layers $\{L_1, L_2, L_3\}$ and the second one containing $\{L_6, L_7\}$.

With these notations defined, the remainder of this section derives a closed expression of the inference throughput achieved by a given partitioning, as a function of the inference (section III-B) and transmission (section III-C) latencies.

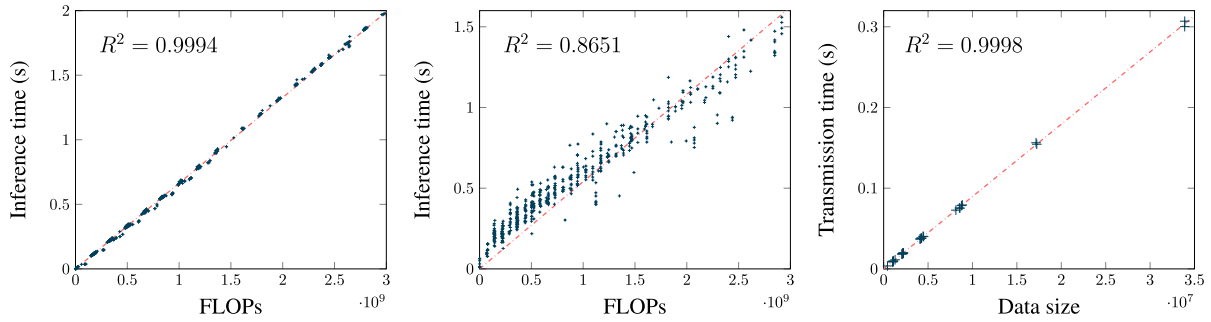
B. INFERENCE LATENCY PREDICTION

Given a partitioning, this section looks at inference latency prediction on an isolated node. The latency induced by the computation of a layer L_i with consumption c_i, to be computed on node N_a, with processing rate η_a is expressed as $T_i^c(N_a) = c_i/\eta_a$. Given a set of layers $\mathcal{L}' \subset \mathcal{L}$ across all threads running on node N_a with processing rate η_a, the inference latency can be expressed as:

$$T^c(N_a) = \eta_a^{-1} \sum_{L_i \in \mathcal{L}'} c_i \tag{1}$$

This expression is a first order approximation of an inference latency estimation based on the number of floating-point operations (FLOPs) required to process the DNN, *i.e.*, the number of multiply-add operations in the model.

Estimation of inference latency on edge devices is complicated by run-time optimizations. Existing solutions such as nn-Meter [18] predict inference latency based on FLOPs. Other solutions rely on a look-up table with pre-computed inference times for latency inference. For example, BRP-NAS [19] uses a pre-trained graph convolutional network



(a) Inference time prediction on CPU (b) Inference time prediction on GPU (c) Transmission time prediction

FIGURE 3. Accuracy of linear modeling for inference and transmission latency. The figures show (i) the dependency between number of FLOPs and inference time on an NVIDIA Jetson Nano, running either on CPU (figure 3a), or GPU (figure 3b), and (ii) the dependency between transmission times and the size of the intermediary vectors sent between layers in a YOLOv2 model in figure 3c. Each figure displays the correlation coefficient R^2 for a linear predictor with zero value intercept.

to predict inference latencies, while taking run-time model optimizations into account.

Inference latency prediction is further complicated by its difference in behavior depending on the underlying hardware. As an example, figure 3 depicts the dependency between FLOPs and inference latency of a YOLOv2[6] [20] model on CPU and GPU.[7] With a linear model as inference latency predictor, it is possible to evaluate the accuracy of such a model by computing the correlation coefficient R^2 of the model on CPU and GPU. The coefficients, displayed in figure 3, depict the difference in model accuracy between CPU and GPU.

In equation 1, it is further assumed that processors use their full capacity, which implies that multi-core processors are modeled as single-core processors with a processing rate equal to the sum of their core processing rates. It is also assumed that the computing resource is shared evenly across threads, *i.e.*, a processor allocates the same proportion of its time to each thread.

Across all nodes, the thread with highest latency sets the inference throughput for the distributed DNN. Omitting transmission latencies, this implies that once this limiting thread is done computing a data frame, it directly starts processing the next one. Other threads will have idle time before processing the next data frame because their inference latency is lower than this maximum latency, as shown on figure 1.

Given a partitioning matrix P, the inference latencies can be expressed as a single vector $\mathbf{T}^c$ of size N where the a-th component is the highest thread inference latency on node N_a, *i.e.*, $\mathbf{T}^c_a = T^c(N_a)$:

$$\mathbf{T}^c = \mathbf{H} \cdot \mathbf{P} \cdot \mathbf{c} \tag{2}$$

where $\mathbf{H}$ is the diagonal matrix of inverse node processing rates $diag(\eta_1^{-1}, \ldots, \eta_N^{-1})$ and $\mathbf{c}$ is the column vector of individual layer consumptions $(c_1, \ldots, c_L)$.

[6]The YOLOv2 model was taken from the ONNX model zoo at https://github.com/onnx/models/

[7]The device used for this experiment is an NVIDIA Jetson Nano.

C. TRANSMISSION LATENCY PREDICTION

The transmission latency can be predicted as follows: the time it takes to send the amount of data $s_{i,j}$ between layers L_i and L_j on edge (N_a, N_b) over a link with measured throughput $\theta_{a,b}$ is $T^t_{i,j}(N_a, N_b) = s_{i,j}/\theta_{a,b}$. The time to achieve data transfers $s_{i,j} \in \mathcal{E}' \subset \mathcal{E}$ over link (N_a, N_b) is:

$$T^t(N_a, N_b) = \theta^{-1}_{a,b} \sum_{s_{i,j} \in \mathcal{E}'} s_{i,j} \tag{3}$$

Similarly to the inference latency prediction, this implies that the links are shared evenly between data transfers from N_a to N_b. The transmission latency prediction relies on the estimation of the link throughput between nodes, more precisely the *goodput*, *i.e.*, the amount of useful data transmitted per second.

Similarly to inference latency estimation, this is a first order approximation, which efficiently offers good prediction results, as shown in figure 3c, with a correlation coefficient of $R^2 = 0.9998$. The drawback of this method is its requirement to saturate the links to get a throughput estimation.

Given a partitioning matrix P, the transmission latency matrix $\mathbf{T}^t$ can be defined as a square matrix of size L, with $\mathbf{T}^t_{a,b} = T^t(N_a, N_b)$:

$$\mathbf{T}^t = \left(\mathbf{P} \cdot \mathbf{S} \cdot \mathbf{P}^\top\right) \circ \Theta \tag{4}$$

where $\circ$ is the term by term product, or Hadamard product, Θ is the inverse throughput matrix, *i.e.*, a square matrix of size N with $\Theta_{a,b} = \theta^{-1}_{a,b}$ and $\mathbf{S}$ is the transmission size matrix, *i.e.*, a square matrix of size L with $\mathbf{S}_{i,j} = s_{i,j}$ if L_i sends data to L_j, 0 otherwise.

IV. DNN PARTITIONING

The objective of DNN partitioning in this study is to maximize the inference throughput, denoted by C, which corresponds to the inverse of the maximum latency between all the different computation and transmission latencies. Given a partitioning $\mathbf{P}$, the inference throughput is defined

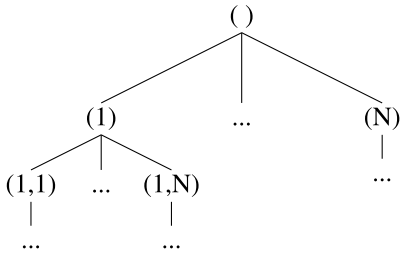


FIGURE 4. Tree representation of the space of all potential DNN partitionings on a physical network of N nodes.

as:

$$C(\mathbf{P}, \mathcal{G}_A, \mathcal{G}) = \left(\max_{i,a,b} \left(\mathbf{T}^c_i, \mathbf{T}^t_{a,b} \right) \right)^{-1} \tag{5}$$

Maximizing the inference throughput amounts to finding the joint min-max between all the terms of $\mathbf{T}^c$ and $\mathbf{T}^t$. This can be defined as a discrete optimization problem:

$$\min_{\mathbf{P}} \max_{i,a,b} \left((\mathbf{H} \cdot \mathbf{P} \cdot \mathbf{c})_i , \left(\left(\mathbf{P} \cdot \mathbf{S} \cdot \mathbf{P}^\top \right) \circ \Theta \right)_{a,b} \right)$$

$$\text{s.t. } \mathbf{J}_{1,N} \cdot \mathbf{P} = \mathbf{J}_{1,L} \tag{6}$$

$$\mathbf{P}_{a,i} \in \{0, 1\} \quad \forall (a, i) \in [\![1, N]\!] \times [\![1, L]\!] \tag{7}$$

with $\mathbf{J}_{i,j}$ being the all-ones matrix of size $i \times j$, conditions (6) and (7) require that each layer be placed on one and only one node.

This optimization problem is a Mixed Integer Non-linear Programming (MINLP) problem. The partitioning matrix can be seen as the one-hot encoded version of a vector of size $(1 \times L)$ with values in $[\![1, N]\!]$. MINLP problems are NP-hard, implying that the complexity of finding an optimal DNN partitioning is $\mathcal{O}(N^L)$ since each of the L layers can be placed on N nodes. There are several heuristics to obtain optimal or sub-optimal solutions to this problem in less time than a brute force algorithm, *e.g.*, Genetic Algorithms (GA) [21], Particle Swarm Optimization (PSO) [22], or Branch and Bound (B&B) [23]. The chosen methodology is an adapted B&B implementation, which is presented in section IV-A.

A. BRANCH AND BOUND ALGORITHM

This section presents the branch and bound (B&B) adaptation to the DNN partitioning optimization problem defined in section IV.

In B&B algorithms, the solution space is represented by a tree. The process consists of eliminating entire branches in the tree based on the evaluation of the score of the root node. This implies the existence of a tree topology which creates a relationship between the score of a node in the tree and the bounds on the scores of all of its leaves.

For this purpose, a partitioning $\mathbf{P}$ is represented as a list $p = (p_i)_{1 \leq i \leq L}$ with $\forall i \in [\![1, L]\!]$, $p_i \in [\![1, N]\!]$ corresponding to the node assigned to layer i. For example, the partitioning matrix in Equation III-A is equivalent to list $p = (1, 1, 1, 2, 2, 1, 1)$, *i.e.*, $\mathbf{P}$ is the one-hot encoded version of p. With this representation, the space of all possible partitionings can be modeled as a tree, as shown in figure 4, with each node being a partial version of the full partitioning list. The root node is an empty list, and at every stage $s \leq N$ of the tree, the nodes are all possible lists of size s. The children of a parent node of size s are all lists of size $s + 1$ beginning with the parent node. Partial lists of size s can represent all partitioning lists which start with a given set of s values.

This tree representation of the solution space favors the use of the B&B algorithm in this study. Assuming p_s is a partial list of size s, the children of the node p_s in the tree topology are all lists which start with p_s, *i.e.*, all partitionings where the first s layers are placed according to p_s. The leaf node below p_s corresponding to p_s padded with the last value of p_s is labeled $\tilde{p_s}$. For example, if $L = 5$ and $p_2 = (2, 1)$, then $\tilde{p_2} = (2, 1, 1, 1, 1)$. Given $T^t_{max} = \max \mathbf{T}^t(\tilde{p_s})$ the maximum value of the transmission latency matrix for partitioning $\tilde{p_s}$, all leaf nodes p below p_s will have a higher maximum transmission latency than $\tilde{p_s}$, *i.e.*, $\forall p$, $\max \mathbf{T}^t(p) \geq T^t_{max}$. This is explained by the fact that given a partial partitioning, any displacement in the other layers will only add data transfers between nodes.

During the process of looking for an optimal partitioning p, which maximizes the inference throughput $C(p, \mathcal{G}_A, \mathcal{G}_N)$, and given a current best achievable throughput best_throughput, the B&B optimization consists of eliminating entire branches of the tree representing the solution space by evaluating transmission times in partial partitionings. This process allows the B&B optimization to quickly eliminate numerous cases compared to a brute force algorithm. For example, if the throughput between nodes is low compared to the compute capacity, *e.g.*, with nodes connected through a low throughput wireless connection, the optimal solution is often to keep all computation on a single node. The advantage of the B&B algorithm is that it terminates after N operations in such simple cases by eliminating all partial partitionings in the first stage of the tree. Conversely, in cases where links have higher throughputs, the complexity of the B&B can be higher since it will be unable to eliminate large branches.

B. BRANCH AND BOUND COMPLEXITY

The overhead caused by the total number of partitions in a real world implementation is neglected in this approach. This added latency is correlated with the total number of partitions. With each partition, the data transfer from Network Interface Controller (NIC) to memory causes an additional latency, related to the PCIe throughput, memory throughput and size of the transferred data. Assuming that the data transfer throughput is limited by the memory throughput, a simplistic evaluation of this latency would change the computation latency in equation 1 to:

$$T^c(N_p) = \nu^{-1} \sum_{(i,j) \in \mathcal{C}} s_{i,j} + \eta_p^{-1} \sum_{L_i \in \mathcal{L}'} c_i \tag{8}$$

In this expression, ν is the memory throughput, *i.e.*, the read/write speed to and from the memory, and $(i, j) \in \mathcal{C}$ are all the layers L_i and L_j such that edge (L_i, L_j) is a split point with associated data transfer $s_{i,j}$.

1) EARLY STOPPING

With an increasing number of partitions, both the performance and inference latency prediction accuracy will decrease. For this reason, and for the increase of complexity in cases with high available network throughput, an early stopping mechanism is added to limit the number of split points in the final partitioning, *i.e.*, B&B will only explore partial partitioning with a maximum of S split points, further limiting the size of the explored tree. The complete B&B algorithm with limited split points is shown in algorithm 1.

Algorithm 1 Branch and Bound Algorithm

Input: $\mathcal{G}_A, \mathcal{G}_N, S$
Output: best_partitioning, best_throughput

```
queue ← {[ ]}
best_throughput ← C([1, ..., 1], G_A, G)
best_partitioning ← [1, ..., 1]
while queue is not empty do
  get p from queue
  generate all children p ‖ {i}, ∀i ∈ [[1, N]]
  compute t_i the maximum transmission time T^t(p ‖ {i})
  for i = 1 to N do
    if t_i > best_throughput^-1 then
      discard p ‖ {i}
    else if t_i ≤ best_throughput^-1 then
      if S(p ‖ {i}) < S then
        add p ‖ {i} to queue
      end if
      if C(p ‖ {i}, G_A, G_N) > best_throughput then
        best_throughput ← C(p ‖ {i}, G_A, G)
        best_partitioning ← p ‖ {i}
      end if
    end if
  end for
end while
```

Limiting the number of split points lowers the complexity of this MINLP problem. For N the number of nodes, and L the number of DNN layers, the number of possible partitionings considered by the algorithm is now lowered to:

$$\sum_{k=0}^{S} \binom{L-1}{k} N\,(N-1)^k \tag{9}$$

instead of $\mathcal{O}(N^L)$ for a brute force algorithm.

V. SIMULATIONS

This section presents simulations to evaluate the impact of the algorithm parameters on the B&B algorithm performance and complexity. The worst case complexity for B&B is described in equation 9, when all of the transmission times computed in the partial partitionings exceed the best achieved time, but the effective B&B complexity varies according to the DNN and network properties. The following variables are explored:

- The relationship between number of nodes N and the maximum allowed number of split points S, which determines the required number of B&B iterations to reach the optimal solution.
- The relationship between link throughput θ and node processing rate η, which determines the existence of a partitioning which improves the inference throughput.

These interactions are evaluated via the following metrics, which can be used to compare the proposed method to the literature:

- The achieved inference throughput, measured in inferences per second.
- The number of B&B iterations to reach the solutions, *i.e.*, the number of explored partitionings in the solutions tree. This metric is used instead of computation time in order to abstract the used device performance. For reference, when running B&B on a desktop computer with an Intel Core i7 processor, it took 17 milliseconds to run through 10^2 iterations, 1.269 seconds to run through 10^4 iterations, and 5 minutes to run through 10^6 iterations.

To isolate each variable contribution, the simulations are run in a homogeneous network scenario, *i.e.*, all nodes have an identical processing rate, and all links have an identical throughput.

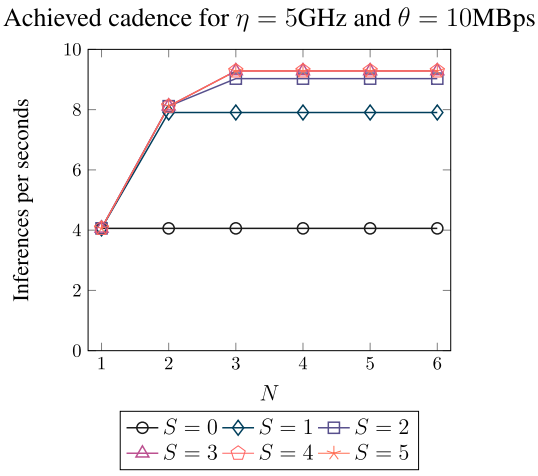
A. NUMBER OF NODES AND SPLIT POINTS

The set-up consists of a homogeneous network with processing rates set at 5GHz (effective processing rate of a standard edge device, *e.g.*, a Raspberry Pi 4[8]), and link throughputs at 10MBps (effective throughputs for connections through 802.11). The number of compute nodes N on the network varies between 1 and 6, and the maximum number of split points S varies from 0 (keeping all computation on a single node) to 5 (for a total of 6 partitions, spread across the network).

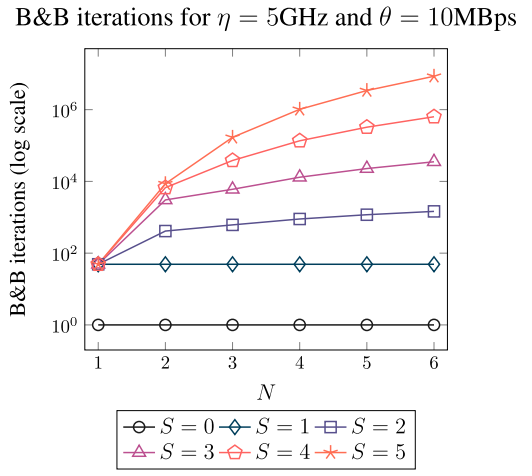
Results are shown in figure 5. It can be observed in figure 5 that, for $N > 2$ nodes in the network, B&B finds partitionings which can multiply the inference throughput by $1.9\times$ to $2.3\times$ the throughput of the unpartitioned solution $S = 0$. By increasing the maximum number of allowed splits, S, the algorithm is able to find better partitionings and improve the inference throughput, at the expense of additional complexity, as indicated in figure 5b.

Figure 5a also illustrates that the best achievable throughput does not improve for maximum number of split points above $S > 3$. With the given values in processing rate and

[8]https://www.raspberrypi.com/products/raspberry-pi-4-model-b



(a) B&B achieved cadence



(b) B&B complexity

FIGURE 5. Impact of the number of nodes *N* and the maximum number of split points *S* on B&B achieved inference throughput (figure 5a), and complexity (figure 5b), for a YOLOv2 model on a homogeneous network.

link throughput, increasing the maximum number of split points does not affect the achieved inference throughput. The main advantage of this observation is that it adds an argument for limiting the B&B maximum split point value S. Since the score stays identical while the complexity increases significantly, choosing $S = 3$ under these conditions both maintains a reasonable complexity and reaches the optimal solution.

This limitation can be explained as follows: with the given DNN and network properties, the optimal placement found on $N = 3$ nodes, and a maximum number of splits $S = 3$, adding a new node to the network, or a possibility for another split point, will not lead to a better partitioning. In a homogeneous network, this implies that any displacement of layers to another node will imply transmission latencies higher than the maximum computing latency of the current partitioning. This leads to a bound on the value of S, after which point the optimal partitioning is the best possible achievable partitioning. This bound can be expressed as:

$$S = \frac{T^c_{\text{mono}}}{T^t_{\text{disp}}} < \frac{\theta \sum_{L_i \in \mathcal{L}} c_i}{s_{\min} \eta} \tag{10}$$

with T^c_{mono} the computing latency when keeping all computation on a single node, T^t_{disp} the transmission latency caused by a layer displacement, η and θ the node processing rate and link throughput values in the homogeneous network, $\sum_{L_i \in \mathcal{L}} c_i$ the sum of all layer consumptions in the DNN, and $s_{\min}$ the minimal inter-layer data transfer size. This expression can be used to define an upper bound on the value of S, and limit the B&B computation time.

For the remainder of this paper, experiments and simulations will be run with a maximum number of splits $S = 3$ to limit the complexity of the algorithm, with near optimal partitioning in the described set-up, which corresponds to a typical edge scenario.

B. PROCESSING RATE AND LINK THROUGHPUT

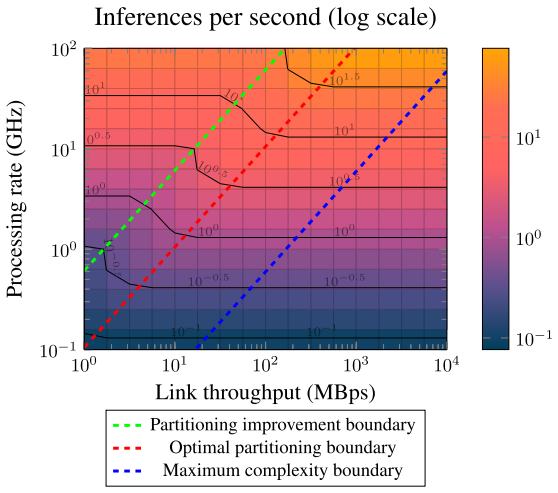
The set-up consists of a standard implementation of YOLOv2 [20], deployed on a network with $N = 4$ compute nodes. B&B is run with a maximum number of split points set to $S = 3$, for node processing rates between 0.1GHz and 100GHz, and link throughputs between 1MBps and 10GBps. The ranges in value for processing rate and link throughput were chosen to cover a wide spectrum of edge scenarios:

- Processing rates between 0.1GHz and 100GHz cover CPUs of systems on a chip, *e.g.*, the Qualcomm Snapdragon suite[9] with processing rates between several 500MHz and 1GHz, and specialized AI embedded systems, *e.g.*, the NVIDIA Jetson TX2 module,[10] with a measured processing rate of 30.7 GHz in the experiments of section VI.
- Link throughputs between 1MBps and 10GBps correspond to link throughputs covering 802.11g connections at several MBps, and Gigabit Ethernet links.

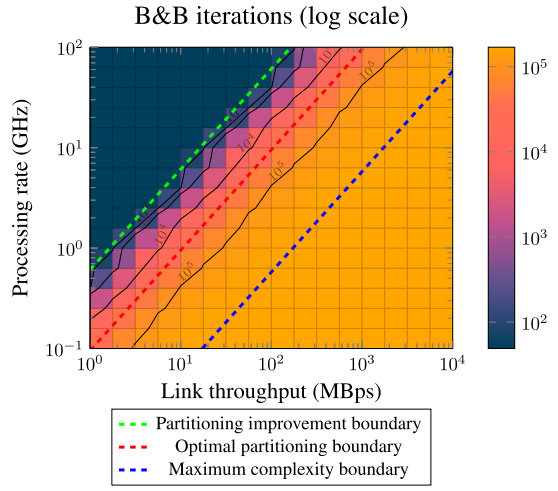
Simulation results are presented in figure 6, composed of two heatmaps, displaying the impact of link throughput and node processing rate, on both the B&B achieved inference throughput (in inferences per second) and complexity (in number of partitionings evaluated by B&B). Level lines are displayed to facilitate interpretation of the figures. Darker colors on the figures correspond to lower inference throughputs (respectively B&B iterations) and lighter colors correspond to higher values, for a set-up of two devices with the corresponding processing rate and link bandwidth value.

Results show that achieved inference throughput and B&B iterations increase with both processing rate and link throughput. This is expected, since faster processors yield faster inferences, and better link throughput creates possibilities for improved partitionings.

[9]https://www.qualcomm.com/snapdragon
[10]https://developer.nvidia.com/embedded/jetson-tx2



(a) B&B achieved inference throughput



(b) B&B complexity

FIGURE 6. Impact of node processing rate and link throughput on B&B achieved inference throughput (figure 6a) and complexity (figure 6b), for a YOLOv2 model on a homogeneous network.

Additionally, these simulations highlight the existence of distinguishable regions in the two heatmaps of figure 6. These regions are separated by three boundaries, corresponding to fixed link throughput to processing rate ratios: the partitioning improvement boundary, the optimal partitioning boundary and the maximum complexity boundary, displayed by the three dotted lines of figure 6. These boundaries are detailed in sections V-B1, V-B2, and V-B3, respectively.

1) PARTITIONING IMPROVEMENT BOUNDARY

The green dotted line separates the top-left region of figures 6a and 6b, where the optimal solution consists of keeping all the computation on the same node, and the bottom-right region, where a non-trivial partitioning that improves inference throughput exists. This boundary corresponds to conditions on the ratio between link throughput and node processing rate under which B&B starts to explore improved solutions in algorithm 1: $\max \mathbf{T}^t \leq \text{best\_throughput}^{-1}$. This is validated under the condition that the smallest data transfer latency between nodes exceeds the unpartitioned computing latency, *i.e.*, the existence of partitionings which improve the unpartitioned inference throughput is subject to:

$$\frac{s_{\min}}{\theta} < \frac{\sum_{L_i \in \mathcal{L}} c_i}{\eta} \qquad i.e., \quad \frac{\theta}{\eta} > \frac{s_{\min}}{\sum_{L_i \in \mathcal{L}} c_i} \tag{11}$$

with η and θ the node processing rate and link throughput values in the homogeneous network, $\sum_{L_i \in \mathcal{L}} c_i$ the sum of all layer consumptions in the DNN, and $s_{\min}$ the minimal inter-layer data transfer size. This expression is a particular case of equation 10, with a number of split-points $S = 1$.

This result is essential, in order to understand DNN partitioning. It defines a criterion on the region where distributing inference can improve the overall performance. In a homogeneous scenario, for every DNN, the link throughput to node processing rate ratio $\frac{\theta}{\eta}$ (which is a property of the network) needs to exceed a deterministic value described in equation 11 (which only depends on properties of the DNN). This boundary is represented by the green dotted line in figure 6 and corresponds to $\frac{\theta}{\eta} \approx 1.64 \times 10^{-3}$ for the YOLOv2 implementation used in this paper.

2) OPTIMAL PARTITIONING BOUNDARY

The red dotted line in figure 6 corresponds to $\frac{\theta}{\eta} \approx 9.52 \times 10^{-3}$ and delimits the point of diminishing returns, which is also the maximum achievable inference throughput for YOLOv2. For higher values of the link throughput to processing rate ratio $\frac{\theta}{\eta}$ (bottom right), the achieved inference throughput remains identical while the number of evaluated partitionings by B&B continues to increase. The additional evaluated partitionings have lower inference throughputs than the optimal solution. This explains why the level lines in figure 6a are horizontal in that region, since the inference throughput only depends on the processing rate. Increasing the ratio above this boundary (for example by changing the links) does not yield better solutions, and increases complexity.

3) MAXIMUM COMPLEXITY BOUNDARY

The blue dotted line represents the boundary of the region in which the number of evaluated partitionings by B&B is maximal, and corresponds to $\frac{\theta}{\eta} \approx 0.168$. On the bottom right part of this boundary, increasing the throughput to node processing rate ratio will not impact the complexity of B&B which has reached the maximum number of evaluations it runs through to reach the optimal solution. Notably, the maximum number of partitionings evaluated by B&B in the worst case scenario is around 1.7×10^5, which remains below the complexity of B&B with limited number of split points $S = 3$ (around 1.9×10^6 iterations in this context), and orders of magnitude below the brute force scenario, which

would need to evaluate $N^L \approx 10^{29}$ partitionings (for the given YOLOv2 implementation with $L = 49$ layers).

C. DISCUSSION

Results of performed simulations allow to identify three boundaries which delimit the scope of validity for DNN partitioning in a homogeneous network. These boundaries depend on the DNN and network properties and delimit the conditions under which a partitioning can improve the unpartitioned inference throughput. The partitioning improvement criterion defines these conditions on the link throughput to processing rate ratio, and can be anticipated prior to the deployment.

While the optimal partitioning boundary and maximum complexity boundary depend on the number of nodes N and the maximum number of split points S, the location of the partitioning improvement boundary is independent of these variables, or the number of layers. The scope of validity of DNN partitioning is independent of network or DNN size, and only depends on the relationship between (i) $\frac{\theta}{\eta}$, which is a property of the network, and (ii) $\frac{s_{min}}{\sum_{L_i \in \mathcal{L}} c_i}$, which is a property of the DNN.

Regarding complexity, it has to be noted that the optimization process is a one-time operation that decides on a partitioning which will remain relevant for long periods, *i.e.*, longer than the order of magnitude of B&B computation times depicted in figure 5. The necessity to recompute a partitioning would only arise if the system experiences persistent changes in the node processing rates or link throughputs. In "healthy" network scenarios scenarios, where faults, or events causing persistent changes, are rare, this paper argues that partitioning computation times of B&B can fit use-cases with one-time deployments.

In more constrained use-cases, the maximum number of split points S can further act as a tuning parameter for the optimization complexity, *e.g.*, if the computation of an optimal partitioning is more frequent, at the expense of the achieved inference throughput.

Compared to works cited in section I-A, B&B can reach optimal solutions with unlimited split points ($S \geq L - 1$), *i.e.*, the best achievable solution in the problem space. The time to reach these solutions increase with α. Nevertheless, it is possible to observe from Figure 5a that this solution can lead to higher inference throughputs than most methods cited in section I-A which are limited to a single point ($S = 1$).

VI. EXPERIMENTS

This section presents experimental results evaluating the accuracy of the model and the achieved inference throughput improvement. Section VI-A describes experiments in homogeneous scenarios with two identical nodes, performed to test the validity the identified boundaries in the simulations presented in section V. Section VI-B describes experiments and presents results in networks with heterogeneous nodes.

A. HOMOGENEOUS NETWORK

Four set-ups are presented in table 2, with details on links, processing rates, and corresponding ratios.

- Set-up 1 corresponds to conditions above the partitioning improvement boundary in figure 6 (section V-B1) where partitioning the DNN does not improve the inference throughput (*i.e.*, the set-up does not fulfill the partitioning improvement criterion of equation 11).
- Set-up 2 corresponds to conditions between the partitioning improvement boundary and the optimal partitioning boundary (section V-B2). DNN partitioning is expected to find partitionings which improve the inference throughput.
- Set-up 3 corresponds to conditions between the optimal partitioning boundary and the maximum complexity boundary (section V-B3). This implies that B&B is expected to find the best achievable partitioning for a given model.
- Set-up 4 corresponds to conditions below the maximum complexity boundary, *i.e.*, the achieved partitioning is optimal and the number of B&B iterations is maximal.

The values in table 2 are experimental and were measured by benchmarking devices and links. Partitions are deployed and run for a YOLOv2 model and a maximum number of splits $S = 3$, for each of the experimental set-ups. The inference is run on a 1280×720 webcam video stream, with 30 frames available per second.

Figure 7 depicts the measured inference throughputs during this process, and compares them to the predicted inference throughputs and to the baseline, *i.e.*, to the throughput achieved by keeping all of the computation on a single node.

The results correspond to the simulations and confirm the existence of the four identified regions of figure 6.

B. HETEROGENEOUS NETWORK

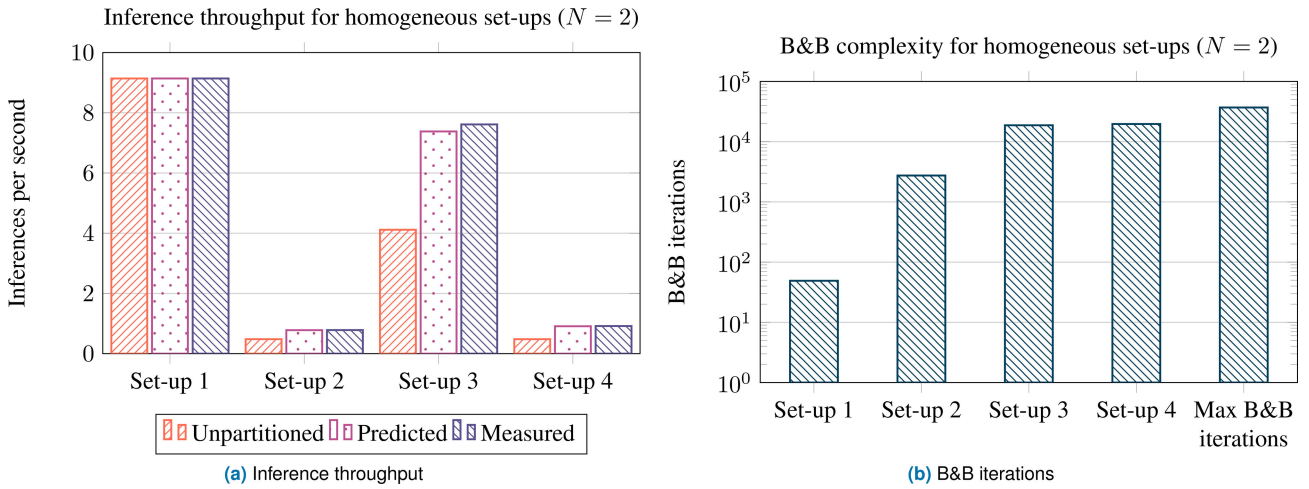
In order to better understand DNN partitioning performance in heterogeneous networks, this section presents an experiment which considers the case of adding a single device with varying capacities to a fixed set-up with a single device. This experiment aims to both show results on simple heterogeneous set-ups, and to illustrate the case of an additional device being added to a network to improve the overall performance, *e.g.*, adding a processor in proximity to a smart camera to increase its inference throughput.

The fixed node[11] has a processing rate $\eta_0 = 14.7$GHz, and figure 8 shows the measured inference throughputs when adding devices with varying processing rates to the network, with two different link throughput values, compared with their predicted values. The figure also depicts bounds on achievable solutions: the lower bound is the unpartitioned inference throughput, *i.e.*, the throughput when placing all computation on the fastest node, and the upper bound corresponds to the throughput of a device with a processing

[11] The fixed processor is an NVIDIA Maxwell GPU (128 CUDA cores).

TABLE 2. Experimental set-ups with measured properties, corresponding link throughput to node processing rate ratios, and associated B&B computation times. Each experimental set-up corresponds to properties in one of the four separate zones identified in figure 6.

Set-up	Processor	Processing rate (η)	Link	Link throughput (θ)	Ratio (α)	B&B time
1	NVIDIA Pascal GPU (256 CUDA cores)	30.7 GHz	Wi-Fi	10 MBps	3.26×10^{-4}	4 ms
2	Quad-core ARM A57 CPU	1.57 GHz	Wi-Fi	10 MBps	6.37×10^{-3}	165 ms
3	NVIDIA Maxwell GPU (128 CUDA cores)	14.7 GHz	Ethernet	1.4 GBps	9.52×10^{-2}	2.231 s
4	Quad-core ARM A57 CPU	1.57 GHz	Ethernet	1.4 GBps	0.98	2.494 s



(a) Inference throughput (b) B&B iterations

FIGURE 7. Achieved inference throughput and complexity for a YOLOv2 model in homogeneous experimental set-ups (table 2). Figure 7a compares the achieved inference throughput with the unpartitioned throughput, and the B&B predicted throughput. Figure 7b compares the effective number of iterations required to compute the partitioning with the maximum number of iterations for $S = 3$ split points.

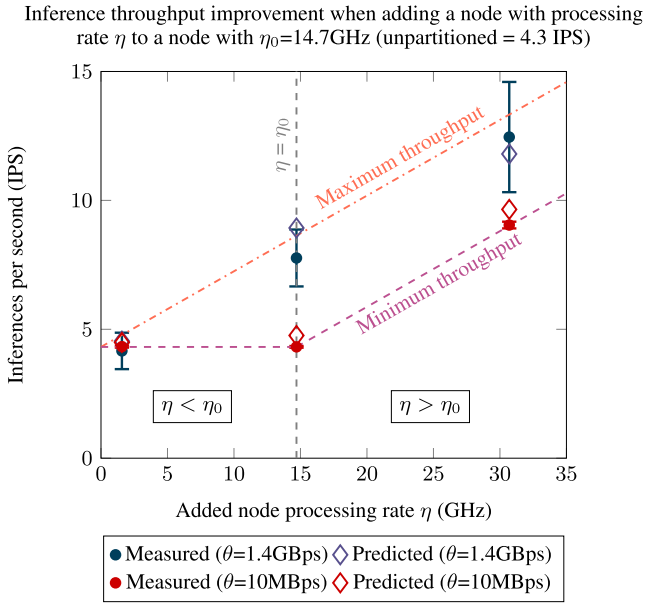


FIGURE 8. Inference throughput improvement when adding devices with varying processing rates η to a device with a processing rate of η_0=14.7GHz. The figure shows measured throughput values with their standard deviation and compares them with the predicted inference throughputs.

rate equivalent to the sum of both processing rates, *i.e.*, corresponding to a perfectly even distribution of the inference workload.

This experiment shows that under good link conditions, *i.e.*, above the partitioning improvement boundary (section 11), the achieved inference throughput can be close to the maximum achievable throughput. Notably, the expression of the partitioning improvement boundary differs from its expression in the homogeneous case (equation 11):

$$\frac{s_{\min}}{\theta} < \frac{\sum_{L_i \in \mathcal{L}} c_i}{\eta_{\max}} \qquad i.e., \quad \theta > \frac{s_{\min}\eta_{\max}}{\sum_{L_i \in \mathcal{L}} c_i} \tag{12}$$

for a heterogeneous case with $N = 2$ nodes, with $\eta_{\max}$ being the maximum processing rate between the two nodes.

This expression implies that for link throughputs below the fixed value $\frac{s_{\min}\eta_{\max}}{\sum_{L_i \in \mathcal{L}} c_i}$, optimal partitioning keeps all computation on the fastest node, *i.e.*, following the minimum throughput line, as is the case when the nodes are connected via a Wi-Fi connection with $\theta = 10$MBps in figure 8. For higher link throughputs, there exist partitionings which improve the inference throughput, and inference throughput values are higher than the lower bound in figure 8, as is the case when the nodes are connected via Ethernet links with $\theta = 1.4$GBps.

VII. RESULTS, SCOPE, AND LIMITATIONS

This section describes key results from simulations and experiments on the DNN partitioning approach, as well as a discussion of their scope, limitations and ways to generalize them.

A. CONDITIONS FOR HOMOGENEOUS NETWORKS

Through simulations (section V) and experiments (section VI), on homogeneous networks, the paper has identified and described conditions, bounds, and closed expressions, for performance and complexity of DNN partitionings. Under the assumption of *homogeneity* (*i.e.*, nodes and links in the underlying network having equivalent capability), these expressions allow to dimension appropriately the underlying network, and predict the partitioning outcome.

- *The partitioning improvement boundary* (section V-B1) describes the conditions under which there are partitionings that improve the monolithic inference throughput. This is extended to the heterogeneous case in equation 12, and allows to estimate the values of link bandwidth and node processing rate, necessary to achieve an inference throughput improvement.
- *The upper bound on the number of split-points* necessary to achieve an optimal solution (equation 10) is the maximal number of split-points worth considering, to minimize the B&B computing time, while accessing maximal inference throughput.
- *The maximal B&B complexity*, with a chosen maximum number of split-points (equation 9).

These results allow to derive, under conditions of perfect distribution of workload over homogeneous nodes and links, an upper bound on the achievable inference throughput of the partitioning strategy:

$$C(\mathbf{P}, \mathcal{G}_A, \mathcal{G}) < \min\left(\frac{\eta \min(N, S)}{\sum_{L_i \in \mathcal{L}} c_i}, \frac{\theta}{s_{\min}}\right) \tag{13}$$

The validity of these expressions is subject to the network homogeneity assumption. For non-homogeneous networks, the derivation of similar performance and complexity bounds is, of course, specific to the characteristics of the considered network.

B. SCOPE AND LIMITATIONS

This section discusses the limitations of this work, and the presented assumptions, to illustrate their scope of applicability.

1) INPUT DATA

The study has used the DNN partitioning framework on video stream data applications. Although the study has not proven its applicability to other data types, there are no assumptions in the modeling restraining this partitioning method from applying to other applications, *e.g.*, audio, text, or telemetry data. The only limiting assumption in this study is that the model used a feed-forward DNN, with data of constant size across requests.

2) OPTIMIZATION PROBLEM DEFINITION

This study focuses on use-cases which require a maximal inference throughput, omitting optimization objectives such as the ones included in the related work of section I-A, *e.g.*, latency, energy consumption, cost, or a combination of these previous metrics. However, the presented assumptions and modeling can be exploited to describe other use-cases. For example, DNN partitioning can cover contexts which:

- jointly optimize several metrics, *e.g.*, throughput, latency, energy consumption, monetary cost, drop rate, node up-time, link usage, etc.
- dynamically adapt what metric to optimize depending on the received data. For example, DNN partitioning application on video streams can choose to optimize throughput to avoid missing detections, and switch to latency when an event occurs in the system, to enable low response times.
- add other constraints to the MINLP problem, *e.g.*, a minimal number of split points, a partial node to layer mapping, *e.g.*, in order to keep sensitive computation on dedicated nodes.

All these assumptions can be expressed as optimization objectives, or constraints in the discrete optimization problem of section IV.

3) LIMITS OF EXPERIMENTAL RESULTS

As described in section VI, the space of possible DNNs and edge networks is too large to explore fully. The experiments have been designed to confirm the boundaries identified in the simulations of section V, and to illustrate a simple heterogeneous use-case. This study also has assumed that the network properties remain fixed over time.

Completely exploring the influence of other parameters on the DNN partitioning performance and complexity, *e.g.*, the number of compute nodes and number of split points in large networks, the complexity of DNN structures, the heterogeneity of layer consumptions and data transfer sizes, or the heterogeneity of link bandwidths and processing rates in large networks — and providing a more thorough understanding of how the system behaves in cases where (i) other processes are dynamically allocated to compute nodes, (ii) links are dynamically used by other processes, or (iii) faults occur on the system (*e.g.*, node failure, routing change, packet drops) — requires additional experiments, following the pattern and methodology of those presented in this paper.

VIII. CONCLUSION

Deploying DL applications in the cloud is convenient because it allows on-demand easy access to computing resources. However, latency or privacy sensitive applications may not be able to exchange data and models with the cloud, while still requiring the same inference throughput to run with good performance. In such cases, DNN partitioning can offer a complementary alternative to hardware acceleration and model optimization to increase inference throughput. This paper has described such an approach to DNN partitioning, which extends previous works by allowing for multiple

split points and multiple threads, and shown to achieve higher inference throughputs than single split point DNN partitioning.

In this context, this paper has quantified the limitations of inference and transmission latency prediction in edge environments. With these assumptions, the DNN partitioning problem is defined as an optimization process, with the objective of maximizing the overall inference throughput. This paper has then introduced a branch and bound algorithm to find optimal DNN partitionings, with a theoretical analysis of its complexity and achieved inference throughput results.

This analysis has led to the identification of the partitioning improvement boundary, a deterministic bound on the network and DNN properties under which a performance improvement can be achieved by partitioning, as well as the cost to compute such solutions, and their expected performance, in a homogeneous network context. This result is essential in understanding DNN partitioning because it defines the scope of validity of this approach, and only depends on (i) the DNN data transfer size to layer consumption ratio, and (ii) the link throughput to processing rate ratio of the underlying network.

Inference throughput accelerations and defined theoretical boundaries are evaluated through experimental setups under varying network conditions. The experimental results also illustrate the behavior of DNN partitioning under heterogeneous network conditions, highlighting the use-case of incrementally adding processing capacity to accelerate inference throughput. These results enable sizing of both DNN and underlying network properties to achieve inference throughput improvements, even prior to the deployment, with deterministic conditions on the necessary link throughputs to enable a maximal inference throughput acceleration through partitioning.

REFERENCES

[1] S. Mochizuki, K. Matsubara, K. Matsumoto, C. L. P. Nguyen, T. Shibayama, K. Iwata, K. Mizumoto, T. Irita, H. Hara, and T. Hattori, "A 197 mW 70 ms-latency full-HD 12-channel video-processing SoC in 16 nm CMOS for in-vehicle information systems," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E100-A, no. 12, pp. 2878–2887, 2017.

[2] Y.-L. Lee, P.-K. Tsung, and M. Wu, "Techology trend of edge AI," in *Proc. Int. Symp. Design, Autom. Test (VLSI-DAT)*, Apr. 2018, pp. 1–2.

[3] Z. Duan, Z. Yang, R. Samoilenko, D. S. Oza, A. Jagadeesan, M. Sun, H. Ye, Z. Xiong, G. Zussman, and Z. Kostic, "Smart city traffic intersection: Impact of video quality and scene complexity on precision and inference," in *Proc. IEEE 23rd Int. Conf. High Perform. Comput. Commun., 7th Int. Conf. Data Sci. Syst., 19th Int. Conf. Smart City, 7th Int. Conf. Dependability Sensor, Cloud Big Data Syst. Appl. (HPCC/DSS/SmartCity/DependSys)*, Dec. 2021, pp. 1521–1528.

[4] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2015, pp. 1135–1143.

[5] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *CoRR*, vol. abs/1412.6115, pp. 1–10, Dec. 2014.

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

[7] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," 2019, *arXiv:1907.08349*.

[8] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, Apr. 2017.

[9] 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.

[10] 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.

[11] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 854–863.

[12] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr./May 2019, pp. 1423–1431.

[13] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware DNN inference in edge computing by exploring DNN model partitioning and inference parallelism," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 3017–3030, May 1, 2021.

[14] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proc. Workshop Mobile Edge Commun.*, Aug. 2018, pp. 31–36.

[15] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "IONN: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proc. ACM Symp. Cloud Comput.*, Oct. 2018, pp. 401–411.

[16] A. G. Ivakhnenko and V. G. Lapa, *Cybernetics and Forecasting Techniques*, vol. 8. New York, NY, USA: Elsevier, 1967.

[17] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ, USA: Prentice-Hall, 2020.

[18] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "nn-Meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *Proc. 19th Annu. Int. Conf. Mobile Syst., Appl., Services*, Jun. 2021, pp. 81–93.

[19] L. Dudziak, T. Chau, M. S. Abdelfattah, R. Lee, H. Kim, and N. D. Lane, "BRP-NAS: Prediction-based NAS using GCNs," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2020, pp. 10480–10490.

[20] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *CoRR*, vol. abs/1612.08242, pp. 1–9, Dec. 2016.

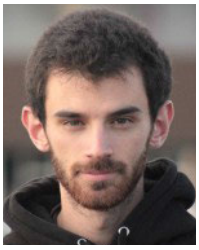
[21] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[22] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, vol. 4, Nov./Dec. 1995, pp. 1942–1948.

[23] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, Jul. 1960. [Online]. Available: http://www.jstor.org/stable/1910129

THOMAS FELTIN received the bachelor's degree in computer engineering from École Polytechnique, Paris, France, and the Master of Science (M.Sc.) degree in advance computing from the Imperial College London, London, U.K. He is currently pursuing the joint Ph.D. degree with École Polytechnique and Cisco Systems, on distributed systems at the edge.

His research interests include the partitioning and distribution of heavy workloads on smart devices at the edge, with a focus on telemetry processing and AI acceleration.



LÉO MARCHÓ received the Master of Science (M.Sc.) degree in computer engineering from École Polytechnique, in 2021. In 2021, he joined the Cisco's Emerging Technologies and Incubation Group to conduct research on AI at the edge. His research interests include computer programming, networks, and optimization. In recent years, he has mainly focused on overcoming optimization challenges when deploying applications at the edge.

JUAN-ANTONIO CORDERO-FUERTES (Member, IEEE) received the M.Sc. (Licenciatura) degree in mathematics and the integrated B.Sc. and M.Sc. (Ingeniería Superior) degree in telecommunication engineering from the Technical University of Catalonia (UPC), Spain, in 2006 and 2007, respectively, and the Ph.D. degree from École Polytechnique, in 2011. His Ph.D. dissertation on the optimization of link-state routing protocols for operation in MANETs and compound autonomous systems. He was a Postdoctoral Researcher with Universitè Catholique de Louvain (UCL), Belgium, and The Hong Kong Polytechnic University, Hong Kong, SAR, China, before joining as a Faculty Member with École Polytechnique, in 2016, where he is currently an Associate Professor. His research and scientific interests include routing protocols and information dissemination algorithms, modeling, analysis and optimization of distributed, adaptive systems in dynamic, and heterogeneous networking scenarios.

FRANK BROCKNERS received the Diploma degree in electrical engineering from Aachen University and the Ph.D. degree in information science from the University of Cologne. He is currently a Distinguished Engineer with the Cisco's Emerging Technologies and Incubation Group, driving software and architecture development for edge platforms, solutions, associated services, and applications. He is involved in several open source projects and a linux foundation networking (LFN) board member. He is an active member of IETF, where his focus is the standardization of observability/OAM solutions (IOAM). He is also a Frequent Speaker at conferences and events, including CiscoLive, where he is recognized as a ''CiscoLive Hall-of-Fame Elite,'' the highest speaker rank.

THOMAS H. CLAUSEN (Senior Member, IEEE) received the M.Sc., Ph.D. (civilingeniør), and cand.polyt degrees from Aalborg University, Denmark.

Since 2004, he has been as a Faculty Member wiith École Polytechnique, France, premiere technical and scientific university, where he was a Professor and he holds the Cisco Endowed ''Internet of Everything'' Academic Chair. At École Polytechnique, he leads the computer networking research group. He has developed and coordinates, the computer networking curriculum, and coordinates the M.Sc.T. programme ''IoT: Innovation and Management.'' He has published more than 100 peer-reviewed academic publications and has authored and edited 24 IETF, standards. He has also consulted for the development of IEEE 802.11s and has contributed the routing portions of the ITU-T G.9903 standard for G3-PLC networks—upon which, e.g., the current SmartGrid & Connected Energy initiatives are built.

Dr. Clausen was named an ''IEEE Computer Society Distinguished Contributor,'' as a part of the 2021 inaugural class.