Mohammed, Thaha; Joe-Wong, Carlee; Babbar, Rohit; Francesco, Mario Di

Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading

Published in:
INFOCOM 2020 - IEEE Conference on Computer Communications

DOI:
10.1109/INFOCOM41043.2020.9155237

Published: 01/07/2020

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Mohammed, T., Joe-Wong, C., Babbar, R., & Francesco, M. D. (2020). Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading. In *INFOCOM 2020 - IEEE Conference on Computer Communications* (pp. 854-863). Article 9155237 (Proceedings - IEEE INFOCOM; Vol. 2020-July). IEEE. https://doi.org/10.1109/INFOCOM41043.2020.9155237

Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading

Thaha Mohammed*, Carlee Joe-Wong†, Rohit Babbar*, and Mario Di Francesco*

*Department of Computer Science
Aalto University

†Electrical and Computer Engineering
Carnegie Mellon University

***Abstract*—Deep neural networks (DNN) are the de-facto solution behind many intelligent applications of today, ranging from machine translation to autonomous driving. DNNs are accurate but resource-intensive, especially for embedded devices such as mobile phones and smart objects in the Internet of Things. To overcome the related resource constraints, DNN inference is generally offloaded to the edge or to the cloud. This is accomplished by partitioning the DNN and distributing computations at the two different ends. However, most of existing solutions simply split the DNN into two parts, one running locally or at the edge, and the other one in the cloud. In contrast, this article proposes a technique to divide a DNN in multiple partitions that can be processed locally by end devices or offloaded to one or multiple powerful nodes, such as in fog networks. The proposed scheme includes both an adaptive DNN partitioning scheme and a distributed algorithm to offload computations based on a matching game approach. Results obtained by using a self-driving car dataset and several DNN benchmarks show that the proposed solution significantly reduces the total latency for DNN inference compared to other distributed approaches and is 2.6 to 4.2 times faster than the state of the art.**

***Index Terms*—DNN inference, task partitioning, task offloading, distributed algorithm, matching game.**

I. Introduction

Intelligent applications are becoming more and more pervasive due to advances in machine learning and artificial intelligence (AI), particularly, in deep learning [1]. Their scope is very broad: it ranges from intelligent assistants (such as Google Now and Amazon Echo) to advanced video analytics in smart cities. As such, they encompass different types of devices, including mobile phones, wearables, and smart objects in the Internet of Things (IoT) [2]. These devices are embedded, thus, also resource-constrained. As a consequence, their processing capabilities are limited, whereas machine learning algorithms are computationally expensive. Still, personal and IoT devices are connected to the Internet; therefore, they can *offload* computing tasks to third-party services running in the cloud. This has been the prevalent approach in the industry as of now, and also a research focus until a few years back [3].

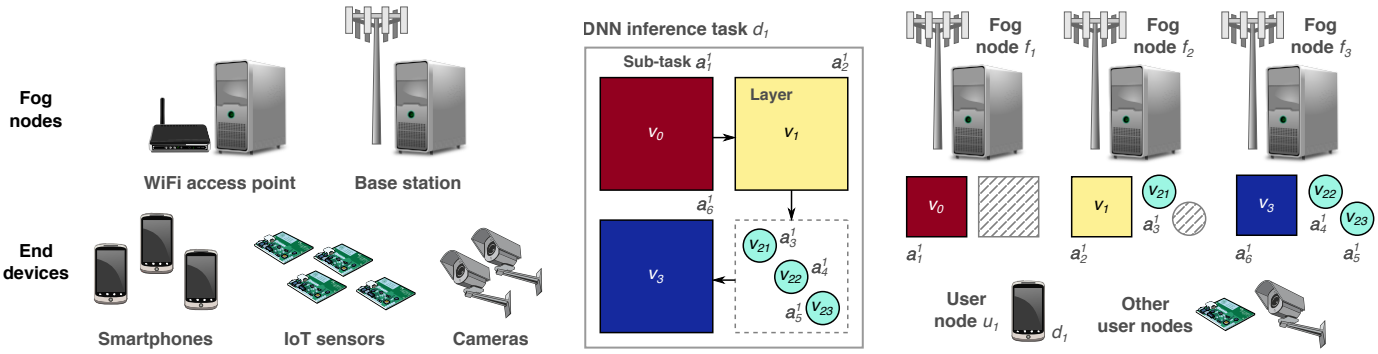
Offloading computations to the cloud involves transferring the source data over the Internet, usually through wireless links. The computational capabilities of embedded devices have been increasing at a much faster pace than radio bandwidth [4]: a typical smartphone of today has 6 to 8 CPU cores running at 2 GHz or more [5], and the latest off-the-shelf embedded IoT devices are equipped with special components (e.g., massively parallel GPUs) to accelerate AI [6]. Such a trend has made processing data locally at the end devices or at the edge of the network more and more efficient.

As a consequence, distributing computations to devices nearby is a better option than offloading them entirely to the cloud [3, 7]. In fact, such an approach reduces the communication overhead while, at the same time, increasing the utilization of computing resources in the network. Accordingly, several solutions have been recently proposed for task offloading [8–11], especially for accelerating[1] deep neural network (DNN) inference (Section VI). A few of them operate only locally [15]; some split DNN computations between the local (or edge) network and the cloud [3, 7]; and others leverage devices in a tiered network architecture [16, 17]. In this context, the main challenge is deciding how to collaboratively *partition* and *distribute* computations under dynamic network conditions.

This article presents DINA (Distributed INference Acceleration), an approach based on matching theory for devices in a (tiered) fog network to carry out distributed DNN inference. Matching theory is a mathematical framework that has originated in economics to describe interactions between two sets of agents with preferences on each other [18]. It has also been applied to wireless networks [19], including for resource allocation in several contexts ranging from device-to-device communications and 5G to the Internet of Things [20–23]. However, to the best of the authors' knowledge, it has not been employed for task offloading in DNN inference acceleration.

The main contributions of this work are the following. First, it proposes a **fine-grained adaptive partitioning** scheme to divide a source DNN in pieces that can be smaller than a single layer (Section III). Partitioning takes into account the specific characteristics of layer types in commonly-used DNNs through an efficient matrix representation that reduces the communication overhead in the network. Second, it presents a **distributed algorithm based on swap-matching** for offloading DNN inference from end devices to nodes in a fog network (Section IV). Such an algorithm is based on a detailed characterization of communication and processing delays, including possible queuing at fog nodes. Finally, **extensive simulations based on a large dataset** and different types of DNNs demonstrate that the proposed solution significantly reduces the total latency for DNN inference compared to other distributed approaches and is 2.6 to 4.2 times faster than the state of the art (Section V).

[1]Related research includes software-driven optimizations, hardware-accelerated DNN computing, as well as federated learning [12–14]. However, these are not considered as they are complementary to the approach proposed here.



(a) (b) (c)

Fig. 1: (a) Reference two-tier fog architecture: end (user) devices and more powerful fog nodes. (b) A DNN inference task consisting of multiple sub-tasks, corresponding to processing either a whole layer or part of it, namely, a partition. (c) Sample DNN offloading of task d_1 in (b) at end user u_1 onto three fog nodes; striped sub-tasks denote those associated with user nodes other than u_1.

II. Modeling DNN Inference in Fog Computing

This section models a fog network wherein DNN tasks are offloaded from user nodes to fog nodes (Fig. 1). The main features of the network are introduced first, followed by a characterization of DNNs and the related computation. Finally, a model for optimal assignment of DNN tasks is introduced. The key parameters of the system are summarized in Table I.

A. Network Model

A fog network (such as the one in Fig. 1a) comprises a set $\mathcal{F} = \{f_1, f_2, \cdots, f_F\}$ of F fog nodes and a set $\mathcal{U} = \{u_1, u_2, \cdots, u_U\}$ of U user nodes (i.e., end devices) distributed in a certain geographical area [24]. Fog nodes are connected to each other and to the cloud through high-speed dedicated links. User nodes communicate with fog nodes over a shared wireless channel with total bandwidth equal to B. In particular, a certain user node can reach all the fog nodes in its transmission range, determined based on Rayleigh channel fading, where the channel gain g_{fu} comprises both shadow fading and distance-dependent path loss according to [25].

User nodes need to run DNN tasks denoted as $\mathcal{D} = \{d_1, d_2, \cdots, d_D\}$. A given task $d \in \mathcal{D}$ might be partitioned into smaller sub-tasks indicated by $\mathcal{A} = \{a_1^d, a_2^d, \cdots, a_A^d\}$. The input and output size of sub-task a are denoted as I_a^s and O_a^s, respectively. The number of cycles to process one element of the input (i.e., the task density) is indicated as c. User nodes process at most one sub-task at a time, while fog nodes can process many due to their higher computational power. Each fog node has a constant CPU-cycle frequency of ν_f.

A certain sub-task can either be executed locally by the user node or offloaded to fog nodes. In the latter case, the input of offloaded sub-tasks need to be transferred from the user to the fog node. A fog node may not execute a sub-task immediately, but can add it to its FIFO execution queue of maximum size Q_f, which is proportional to its computing power.

B. DNN Model

A complete DNN associated with d is modeled as a Directed Acyclic Graph (DAG) $\mathcal{G} = (\mathcal{E}, \mathcal{V})$ where $\mathcal{V} = \{v_0, v_1, v_2, \cdots, v_n, v_{n+1}\}$ is the set of vertices representing the layers of the DNN; particularly, v_0 and v_{n+1} denote the input layer and the output layers (respectively). An edge $(v_i, v_j) \in \mathcal{E}$ represents the dependency of node v_j on v_i. Layer v_i needs to be computed first and the resulting weights are passed as input to layer v_j for later processing. The DNN model of d is pre-loaded at all nodes in the network.

Vertices (layers) are not atomic and can be subdivided into multiple sub-vertices (sub-layers), as illustrated in Fig. 1b. In particular, any vertex v_i – i.e., excluding the input and output layers – can be split such that $v_i = \{v_{i1}, v_{i2}, \cdots v_{in}\}$. The attributes of the vertices are matrices that either denote the weights or the feature map at each layer of the DNN depending on their type. These attributes are represented as $\boldsymbol{R} \in \mathbb{R}^{m \times n}$ where m and n denotes the size of a given vertex v_l and the subsequent one v_k. A vertex with multiple outgoing edges has several attributes $\boldsymbol{R} \in \mathcal{R}$. As a result of partitioning, a matrix $\boldsymbol{R}$ is divided into sub-matrices $\boldsymbol{R}_{ij}$, with $1 \leq i \leq m$ and $1 \leq j \leq n$.

C. Computation Model

The computation time of a DNN depends on the type of its layers. This work considers multi-channel convolution, feed-forward, and activation layers.

The convolution operation at a certain layer across multiple channels [26] is calculated as:

$$\mathbf{Z}_{i,j,k} = \sum_{l} \sum_{m} \sum_{n} \mathbf{I}_{l,j+m-1,k+n-1} \cdot \mathbf{K}_{i,l,m,n} \qquad (1)$$

where $\mathbf{K}$ is the 4D kernel tensor and $\mathbf{I}$ is the 3D input consisting of the input data. Here, $K_{i,j,k,l} \in \mathbf{K}$, $\forall i, j, k, l \in \mathbb{Z}$ is the connection weight between an element in channel i of the layer output and an element in channel j of the input with an offset of k rows and l columns. Moreover, $I_{i,j,k} \in \mathbf{I}$, $\forall i, j, k \in \mathbb{Z}$ gives the input data value within channel i at row j and column k. Assuming that convolution is implemented as a sliding window, the total number of floating point operations for a single layer is given by

$$C_f = 2 \cdot I_k^h \cdot I_l^w \cdot (c_{in} \cdot K_i^w \cdot K_j^h + 1) \cdot c_{out}$$

TABLE I: Summary of used notation

Symbol	Description
$\mathcal{F}$	Set of fog nodes
$\mathcal{U}$	Set of user nodes (end devices)
$\mathcal{D}$	Set of DNN inference tasks
$\mathcal{A}$	Set of DNN layers / partitions
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	DNN graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$
$\mathcal{R}, \boldsymbol{R}$	Set of weight matrices (feature maps) and its element $\boldsymbol{R}$
ν_f	Computing power of fog node f
Q_f	Queue length of fog node f
C_f	Number of FLOPs for a convolution layer
F_f	Number of FLOPs for a fully-connected layer
T^c	Time for computing a convolution layer
T^f	Time for computing a fully-connected layer
r_{uf}	Data rate of user node u towards fog node f
I_a^s	Size of the input task a in bits
O_a^s	Size of the output resulting from processing task a in bits
T_{fa}	Total execution time for task a at fog node f
T_{fa}^{exe}	Execution time for task a at fog node f
T_{fa}^{tr}	Transmission time for task a to fog node f
T_{fa}^{que}	Queuing time for task a at fog node f
θ	Maximum fog nodes for offloading
x_{fa}	Binary variable denoting task a assigned to fog node f
τ_a	Time threshold for task a
Γ	Target transmission rate threshold
δ	Edge delay
c	Task processing density of fog nodes
$\overline{s}_{fa}$	Average service rate at fog node f for task a

where: I_l^h, I_k^w, K_i^w, and K_j^h are the height and width of the input feature map and the kernel, respectively; c_{in} and c_{out} are the number of channels in the input and output feature maps. Thus, the execution time for a single convolution layer is $T^c = (C_f \cdot c)/\nu_f$.

Instead, the number of floating point operations for a (fully-connected) feed-forward layer is $F_f = (I \cdot O + O \cdot (I-1)) = (2I-1)O$, where I and O are the input and output dimensions. The related computation time is $T^f = (F_f \cdot c)/\nu_f$.

The activation layer is assumed to be a rectified linear unit, which simply computes $f(x) = \max(0, x)$. The related execution time is not considered part of the total time, since it is negligible compared to convolutions and dot products. Accordingly, the time T_{fa}^{exe} for computing a deep learning partition a is $T_{fa}^{exe} = T^c$ for a convolution layer and $T_{fa}^{exe} = T^f$ for a fully-connected layer.

Next, the time needed to exchange inputs and outputs between a user node and a fog node are derived. Such a time depends on the transmission rate of user nodes, which is computed through the Shannon-Hartley theorem [27]:

$$r_{uf} = \beta \log_2 \left(1 + \frac{p_u g_{fu}}{\sigma^2 + \sum_{u' \neq u \in \mathcal{U}} p_{fu'} g_{fu'}} \right) \tag{2}$$

where: β is the total bandwidth assigned to user node u; p_u is the transmit power of user node u; g_{fu} is the channel gain between the fog node f and user node u; $p_{u'}$ and $g_{fu'}$ are the transmit power and the channel gain of interfering node u', respectively. Consequently, the time taken by a user node to send the DNN partition a to fog node f is $T_{fa}^{tr} = (I_a^s + O_a^s)/r_{uf}$. As for the downlink, fog nodes need to send the intermediate outputs to users only for the local tasks that require that as input. Moreover, intermediate outputs are generally much smaller than the input [3, 7]; as a consequence, the corresponding time is assumed to be small and is represented as an edge delay δ, as in [28].

Recall that offloaded tasks may not be executed immediately at the fog node, hence, they are delayed due to queuing time

$$T_{fa}^{que} = \sum_{a_i \in Q_f} \frac{\text{flops}(a_i) \cdot c}{\nu_f}$$

where $\text{flops}(a_i)$ is the number of floating point operations for sub-task a_i in the execution queue $i \in Q_f$ at fog node f.

D. Problem Formulation

Distributing DNN inference in a fog network aims at minimizing the total execution time of a given DNN by partitioning and offloading it to one or more fog nodes (Fig. 1c). For the sake of simplicity, the following considers first the case where partitioned tasks are given as input to the problem; this assumption is relaxed at the end of the section.

The offloading problem is defined in terms of the binary variable $x_{fa} \in \{0, 1\}$ expressing whether sub-task a is offloaded to fog node f (i.e., $x_{fa} = 1$) or not (i.e., $x_{fa} = 0$). Then the total execution time for a sub-task a offloaded by user node u to a fog node f is:

$$T_{fa} = T_{fa}^{tr} + T_{fa}^{exe} + T_{fa}^{que} + \delta \tag{3}$$

where: T_{fa}^{tr} is the time taken for transmitting the input of a to f; T_{fa}^{exe} is the time taken for computing a at f; T_{fa}^{que} is the sub-task waiting time at a; and δ is the edge delay.

$$\min_{x_{fa}} \sum_{f \in \mathcal{F}} \sum_{a \in \mathcal{A}} T_{fa} \cdot x_{fa} \tag{4a}$$

$$\text{s.t.} \quad \sum_{f \in \mathcal{F}} x_{fa} \leq \theta, \forall u \in \mathcal{U} \tag{4b}$$

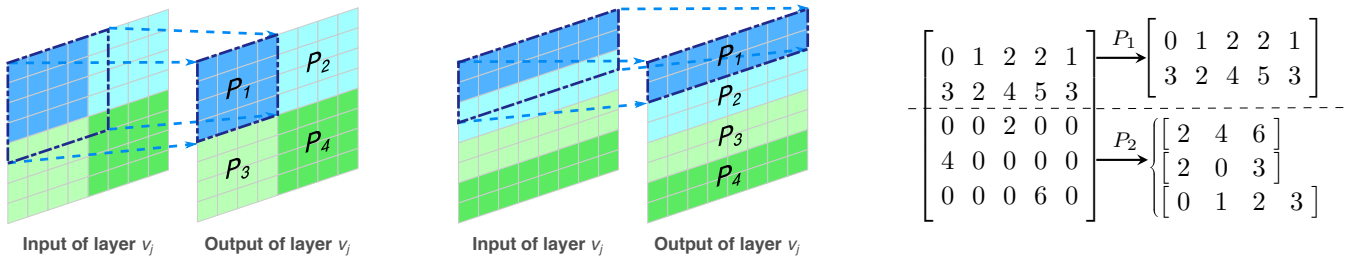
$$\frac{p_u g_{fu}}{\sigma^2 + \sum_{u' \neq u \in \mathcal{U}} p_{fu'} g_{fu'}} \geq x_{fa} \Gamma, \forall u \in \mathcal{U} \tag{4c}$$

$$T_{fa} \leq \tau_a, \forall a \in \mathcal{A} \tag{4d}$$

$$\max_{a \in \mathcal{A}^*} T_{fa} < \sum_{a \in \mathcal{A}^*} \tau_a, \forall \mathcal{A}^* \in \text{P}(\mathcal{A}), \forall u \in \mathcal{U} \tag{4e}$$

The meaning of the constraints in the optimization problem is explained next. Eq. (4b) signifies that task d is offloaded to at most θ fog nodes at a given time. Eq. (4c) indicates that the rate of transmission should never be less than a target value Γ. Eq. (4d) indicates that the total time taken to compute the task at a fog node should not take more than the time τ_a for executing the same task locally (i.e., without offloading). Finally, Eq. (4e) states that parallel execution of tasks at fog nodes should take less time than processing the same tasks locally as a sequence. This is expressed via $\text{P}(\mathcal{A})$, which denotes all possible sets of tasks that can be executed in parallel by up to θ fog nodes.

The formulation above is a non-linear programming problem with a non-convex cost function, which directly follows from Eq. (2). Finding an optimal solution to such a problem is computationally complex [29]. Note that the problem assumes that the partition $\mathcal{A}$ is given. In practice, all possible partitions



(a) (b) (c)

Fig. 2: Comparison of redundant data transmitted during convolution for (a) grid-based 2D partition and (b) segment-based 1D partition. (c) Hybrid representation of the matrix: P_1 is a dense representation and P_2 is a sparse representation.

of the source task into sub-tasks should be considered for optimal offloading. This implies that finding a solution is hard.

Consequently, the problem is addressed next by considering two separate sub-problems: adaptive partitioning of the DNN layers and their distribution (i.e., offloading) to fog nodes.

III. Adaptive DNN Partitioning

This section focuses on the problem of partitioning a given DNN into (sub)layers so that they can be offloaded to fog nodes. First, it addresses how to efficiently represent DNN inputs, so as to reduce their storage requirements as well as their transmission time. It then presents a DNN partitioning algorithm that adapts to the current state of the network.

A. DNN Layer Representation

As before, the discussion below distinguishes between convolutional and fully-connected layers.

The way layers are partitioned affects the communication overhead, especially for convolutional neural networks that need to process data across partition boundaries. For instance, Fig. 2a and Fig. 2b show a 2D (i.e., grid-based) and a 1D (i.e., segment based) partitioning of an 8×8 DNN layer (respectively) into 4 partitions of the same size, each to be offloaded to a fog node. For a 2×2 kernel, computing the output feature map of partition P_1 in Fig. 2a requires transmitting nine extra values with grid-based partitioning: four from P_2, four from P_3, and one from P_4. In contrast, segment-based partitioning of P_1 in Fig. 2b requires eight extra values, only from a single partition (P_2). Accordingly, the approach in this work relies on partitioning convolutional layers into segments along the largest dimension of the input [15].

The fully-connected layers considered here also include state-of-the-art pruned network layers which are sparser than "traditional" fully-connected[2] layers [30]. Most computing time in a fully-connected layer involves matrix-vector multiplication. The sparsity of a matrix and a suitable data structure determine the computational cost as well as the transmission time [31]. Adaptive partitioning here employs the Compressed Sparse Row (CSR) storage scheme [32]. The size of a matrix in CSR format is $12 \cdot \hat{z} + 4 \cdot (m+1)$ bytes, where $\hat{z}$ is the total number of non-zero elements and m the number of rows; in contrast, a dense matrix requires $8 \cdot m \cdot n$ bytes. Accordingly, the CSR storage scheme is applied if $\hat{\theta} < \hat{z}/(m \cdot n)$, where $\hat{\theta}$ is a user-defined threshold between $[0, 1]$ and m, n are the dimensions of the partition. Fig. 2c illustrates a source matrix divided into a dense partition P_1 and a sparse partition P_2 encoded in CSR format. The example shows the advantage of the latter choice, as storing P_2 only requires 52 bytes as opposed to the 120 bytes of a dense representation.

B. DNN Partitioning Algorithm

The partitioning algorithm itself, called DINA Partitioning (DINA-P), is presented next. DINA-P relies on the concept of *utility function* as a measure on how valuable is partitioning a certain task. Specifically, the service utility of fog node f for executing task a of user node u is:

$$\varphi_{fa} = x_{fa}(t) \left(\tau_a - T^{tr}_{fa} - T^{exe}_{fa} - \overline{T}^{que}_{fa} \right) \tag{5}$$

where τ_a is the delay threshold for the task $a \in \mathcal{A}$ and $\overline{T}^{que}_{fa} = \sum_{a_i \in Q_f} (\text{flops}(a_i)/\overline{s}_{fa_i}(t))$ is the average queuing latency. In particular, $\overline{s}_{fa_i}$ is the service rate at fog node f for computing task a_i received from user node u, which can be estimated based on an exponentially weighted moving average:

$$\overline{s}_{fa_i} = \alpha(t)\overline{s}_{fa_i}(t) + (1 - \alpha(s))\overline{s}_{fa_i}(t-1) \tag{6}$$

where α is a learning parameter and t the time of calculating the estimate. The rationale behind this choice is the following: partitioning should occur only if the task is initially[3] offloaded to node f (i.e., $x_{fa} = 1$); it is as valuable as its computation time (including overheads) is lower than the time threshold τ_a [recall Eq. (3)].

Algorithm 1 describes DINA-P as executed at node u upon arrival of task d. After initializing the variables, all parallel paths in the source DNN are considered (line 1). For each fog neighbor f (line 2), the user node calculates the partitioning ratio ρ based on the corresponding service utilities and computing capabilities (line 3).

Such a ratio is applied to divide the task (i.e., matrix) into partitions (i.e., submatrices) depending on the type of layer, convolutional (lines 4–6) or fully-connected (lines 7–11), according to the discussion in Section III-A. The ratio ρ

[2]This article refers to either dense or sparser layers after pruning as fully-connected, as this is also the most widely used terminology in the literature.

[3]Such an initial assignment can be randomly obtained (see Section IV-B).

Algorithm 1: Adaptive DNN partitioning (DINA-P)

Input : p: convolution kernel size; φ_{fa}: utility of f; $|f_n|$: number of fog neighbors f of $u \in \mathcal{U}$; $\mathcal{G}$: DAG for DNN inference task d; $\boldsymbol{R}_a^{m \times n}$: matrix associated with subtask a, $\forall a \in \mathcal{A}$; c: compute power of f.
Output : DNN partitions $P = \{P_1, P_2, P_3, \ldots\}$
Init : $P \leftarrow \varnothing$, $c_0 = 0$, $w \leftarrow \max(m, n)$

forall a belonging to parallel paths in $\mathcal{G}$
  for $i \in [1, |f_n|]$
    $\rho_i \leftarrow \dfrac{\sum_{j=0}^{i-1} (c_j/\varphi_{fa})}{\sum_{j=0}^{|f_n|} (c_j/\varphi_{fa})}$
    if convolutional layer
      $\omega_i \leftarrow \lfloor \rho_i \cdot (w - (p-1)) \rfloor$
      $P_i \leftarrow P_i \cup \boldsymbol{R}_a[\omega_{i-1}][\omega_i + (p-1)]$
    else `// fully-connected layer`
      if $\hat{\theta} < \hat{z}/(m \cdot n)$ **then** store $\hat{P}$ in CSR format
      $\omega_i \leftarrow \lfloor \rho_i \cdot w \rfloor$
      $\hat{P} = \boldsymbol{R}_a[\omega_{i-1}][\omega_i]$
      $P_i \leftarrow P_i \cup \hat{P}$
    $P \leftarrow P \cup P_i$

adaptively partitions the layers into multiple partitions p based on the network conditions to provide maximum utility for both the fog nodes and the user nodes.

Note that the DNN partitioning obtained as described above is adaptive, as DINA-P leverages time-varying service utilities that express network conditions dynamically.

IV. DNN Inference Offloading

Once partitions are derived, they can be offloaded from user nodes to fog nodes. Accordingly, this section presents DINA Offloading (DINA-O), a distributed solution based on matching theory to solve the optimization problem in Section II-D. The rest of this section introduces first matching games and their characterization for the system model considered here. It then details DINA-O and presents an analysis of its properties.

A. Matching Games with Externalities

Matching theory is a mathematical framework in economics that models interactions between two sets of selfish agents competing within a set to match agents in the other set [18]. The related *two-sided matching problems* have been addressed through game-theoretic approaches with the goal of achieving *stability* [33, 34]: breaking a stable matching provides no advantages to any of the agents in the system. Two-sided matching problems are defined by the constraints on the nature of the matching itself, for instance, if an agent in one set can be matched to exactly a single agent in the other set (one-to-one) or to more than one (one-to-many) [35].

Based on the characterization in Section II, a single DNN task d can be offloaded to multiple fog nodes; whereas a single fog node can be associated with multiple user nodes to run their (sub)tasks. Therefore, a many-to-many matching is considered in this work, as formally stated below.

Definition 1 (Many-to-many matching)**.** *Given two disjoint sets of user nodes and fog nodes $(\mathcal{U}, \mathcal{F})$, a many-to-many matching μ is defined as a mapping from the set $\mathcal{U} \cup \mathcal{F}$ into the set of all subsets of $\mathcal{A} \cup \mathcal{F}$ such that $\forall u \in \mathcal{U}$ and $\forall f \in \mathcal{F}$:*

1) $\mu(u) \subset \mathcal{F}$, $\forall u \in \mathcal{U}$ *and* $\mu(f) \subset \mathcal{U}$, $\forall f \in \mathcal{F}$
2) $|\mu(u)| \leq \theta$, $|\mu(f)| \leq U$
3) $f \in \mu(u) \Leftrightarrow u \in \mu(f)$

The first condition simply indicates that one user node is matched to multiple fog nodes and vice versa. The second condition bounds the number of agents belonging to the matching: a fog node can accept tasks from the maximum number of user nodes U in the network, while each user node can be associated with at most θ fog nodes (according to Section II-D). The last condition states that if user node u is matched to fog node f, then the reverse should also hold.

Matching is obtained according to *preference relations*, generally indicated with the $\succ$ symbol, which express the ranking agents in one set have for all the agents in the other set. In this context, $\succ_u$ is the preference relation of user u, while $\succ_f$ is the preference relation of fog node f. If user u_1 prefers fog node f_1 over fog node f_2, then the relation is indicated as $f_1 \succ_{u_1} f_2$. Similarly, if fog node f_1 prefers user node u_1 over user node u_2, then the relation is denoted as $u_1 \succ_{f_1} u_2$.

Preferences are derived by means of utility functions, similar to partitioning. Also here, the utility of fog node f represents the total benefit it obtains by executing task a of user u (namely, when user u offloads task a to f), as expressed by Eq. (5). Accordingly, it is:

$$a \succ_f a' \Leftrightarrow \varphi_{fa} \geq \varphi_{fa'} \tag{7}$$

Instead, the utility of user node u for matching with a fog node f is inversely proportional to the total execution time:

$$\varphi_{uf} = \frac{1}{T_{fa}} \tag{8}$$

In terms of user preference, it is:

$$f \succ_u f' \Leftrightarrow T_{fa} \leq T_{f'a} = f \succ_u f' \Leftrightarrow \varphi_{uf} \geq \varphi_{uf'} \tag{9}$$

User and fog nodes independently rank each other according to the utility functions in Eq. (8) and Eq. (5). To do so, they exchange information about their current state – including queuing times and transmission rates – with each other once a user needs to run a DNN inference task.

It is worth noting that the preferences defined above for fog nodes [Eq. (7)] and user nodes [Eq. (9)] depend on each other; particularly, a matched user is affected by the matching of other users to the same fog node. In matching theory, such kind of dependence is referred to as *externality* [36]. Unfortunately, the basic results of matching theory – in particular, stability – do not apply to matching problems with externalities [37] due to the dynamic nature of preferences. To address this issue, the following leverages the concept of two-side exchange stability [38], which assumes that no user node can remain unmatched, thereby allowing to swap user tasks to fog nodes. Some preliminary definitions are introduced next, following the notation in [20].

Definition 2 (Swap matching)**.** *Let a matching μ be given as well as pairs (f,u) and (f',u'), such that $u, u' \in \mathcal{U}$ and $f, f' \in \mathcal{F}$ with $f \in \mu(u), f' \in \mu(u'), f \notin \mu(u')$ and $f' \notin \mu(u)$. A swap matching $\mu_{u,f}^{u',f'}$ is:*

$$\mu_{u,f}^{u',f'} = \{\mu \setminus \{(u,\mu(u)),(u',\mu(u'))\}\} \cup \{(u,\{\{\mu(u) \setminus f\} \cup f'\}),(u',\{\{\mu(u') \setminus f'\} \cup f\})\}$$

In other words, a swap matching enables any two user nodes to swap one of their fog nodes as long as the matching of other users and other fog nodes remains the same.

Definition 3 (Swap-blocking pair)**.** *Given a matching μ and two user nodes u and u', the pair (u,u') is swap-blocking if and only if it satisfies the following conditions:*

1) $\forall y \in \{u, u', f, f'\}$, *such that* $\Phi_y(\mu_{u,f}^{u',f'}) \geq \Phi_y(\mu)$
2) $\exists y \in \{u, u', f, f'\}$, $\Phi_y(\mu_{u,f}^{u',f'}) > \Phi_y(\mu)$

where $\Phi_y(\mu) = \varphi_{y\mu(y)}$ is the utility function of y for μ.

The first condition states that the utility function of all user nodes u, u' and fog nodes f, f' should not decrease after the swap. Similarly, the second condition signifies that the utility of at least one node should improve following the swap. Clearly, both the user nodes and the fog nodes in the swap-blocking pairs should approve such a swap, and swaps are sought among these pairs.

Definition 4 (Two-sided exchange stability)**.** *A matching μ is said to be two-sided exchange stable if and if only if a swap-blocking pair does not exist.*

Equivalently, a matching μ is two-sided exchange stable if no user u or fog node f prefers another fog node f' or user node u' with respect to its current matching

B. DNN Offloading Algorithm

DINA Offloading (DINA-O) leverages the concept of two-sided exchange stability to find a matching by means of swapping node pairs, as detailed in Algorithm 2.

DINA-O includes two phases: *initialization* and *swap matching*. In the initialization phase (lines 1–6), each user node u first discovers its neighboring fog nodes f. All nodes then calculate their signal to noise interference ratio to derive the rate in Eq. (2) and exchange their current operating parameters [i.e., in relation to Eq. (3)]. At this point, both user and fog nodes are able to calculate their utilities; based on these, they construct their preference relations. Finally, an initial matching that satisfies the constraints in Eq. (4b) – (4d) is derived through a random assignment of user nodes to fog nodes [39]. Based on such an assignment, DNN partitioning is performed with DINA-P (i.e., Algorithm 1).

In the swap matching phase (lines 7–27), the matched user-fog node pair performs a swap matching if there is a swap-blocking pair and update their utilities. At each iteration, a user node sends a request to its preferred fog nodes if they are not yet matched to them. Then fog nodes calculate the new utilities for task a from user u and accept the proposal only if their utility is improved by swap matching. If the proposal is rejected, the user nodes makes a request to the next preferred fog node for swapping. This phase ends when the matching at a certain iteration is the same as in the previous one; consequently, the final matching is obtained and offloading occurs accordingly.

Algorithm 2: DNN Inference Offloading (DINA-O)

Input : A set of fog nodes $f \in \mathcal{F}$ and UNs $u \in \mathcal{U}$
Output : A two-sided exchange-stable matching μ
`// Phase 1: Initialization`
1 Each $u \in \mathcal{U}$ discover its neighboring fog nodes $f \in \mathcal{F}$
2 All nodes $\forall u \in \mathcal{U}, \forall f \in \mathcal{F}$ calculate r_{uf} with Eq. (2)
3 $\forall f \in \mathcal{F}$ creates a preference list with Eq. (5)
4 $\forall u \in \mathcal{U}$ creates a preference list with Eq. (8)
5 Randomly match $(u,f), \forall f \in \mathcal{F}, u \in \mathcal{U}$ such that the constraints in Eq. (4b) – (4d) are satisfied
6 Execute Algorithm 1 to partition DNN tasks $d \in \mathcal{D}$
`// Phase 2: Swap matching`
7 **while** $\exists \mu_{u,f}^{u',f'} : (f', \mu_{u,f}^{u',f'}) \succ_u (f,\mu), (u', \mu_{u,f}^{u',f'}) \succ_f (u,\mu), (f, \mu_{u,f}^{u',f'}) \succ_{u'} (f',\mu), (u, \mu_{u,f}^{u',f'}) \succ_{f'} (u',\mu)$
8 Update φ_{fu} and φ_{fu} based on μ
9 Sort fog nodes $f \in \mathcal{F}$ based on preference $\succ_u$
10 Sort user nodes $u \in \mathcal{U}$ based on preference $\succ_f$
11 **if** $\mu_{u,f} = \varnothing$ `// There is an unmatched item o`
12 u sends proposal to most preferred f
13 f computes $\varphi_{fu}\left(\mu_{u,o}^{o,f}\right)$
14 **if** $(u, \mu_{u,o}^{o,f}) \succ_f (u,\mu)$ and Eq. (4b)–(4d) hold
15 Accept proposal, $\mu \leftarrow \mu_{u,f}^{u',f'}$
16 $\Lambda_f \leftarrow \Lambda_f \cup \{u\}$, $\Lambda_u \leftarrow \Lambda_u \cup \{f\}$
17 **else** Reject proposal and keep matching μ
18 **if** $(f', \mu_{u,f}^{u',f'}) \succ_u (f,\mu)$ and $(f, \mu_{u,f}^{u',f'}) \succ_{u'} (f',\mu)$
19 u sends proposal to f' and u' to f
20 f, f' compute $\varphi_{f'u}(\mu_{u,f}^{u',f'}), \varphi_{fu'}(\mu_{u,f}^{u',f'})$
21 **if** $(u', \mu_{u,f}^{u'f'}) \succ_{f'} (u,\mu)$ and Eq. (4b)–(4d) hold
22 Accept proposal, $\mu \leftarrow \mu_{u,f}^{u',f'}$
23 $\Lambda_{f'}, \Lambda_u \leftarrow \Lambda_{f'} \setminus \{u'\} \cup \{u\}, \Lambda_u \setminus \{f\} \cup \{f'\}$
24 **else if** $(u, \mu_{u,f}^{u'f'}) \succ_f (u',\mu)$ and Eq. (4b)–(4d) hold
25 Accept proposal, $\mu \leftarrow \mu_{u,f}^{u',f'}$
26 $\Lambda_f, \Lambda_{u'} \leftarrow \Lambda_f \setminus \{u\} \cup \{u'\}$, $\Lambda_{u'} \setminus \{f'\} \cup \{f\}$
27 **else** Reject proposal and keep matching μ

C. Analysis of DINA-O

The following proves stability and convergence of DINA-O.

Proposition 1 (Stability)**.** *Let us assume that DINA-O converges to a matching μ^*. Then, μ^* is a two-sided exchange-stable matching.*

Proof: Let us assume that a swap-blocking pair does exist in the matching μ^* obtained upon convergence. Accordingly, DINA-O proceeds with a new iteration as the condition on line 7 of Algorithm 2 is not satisfied. However, this is in contrast with the assumption that DINA-O has converged to μ^*. By contradiction, a swap-blocking pair does not exist, which implies that μ^* is a two-sided exchange-stable matching by Definition 4. ■

Proposition 2 (Convergence)**.** *DINA-O converges within a finite number of iterations as well as swap matchings.*

TABLE II: Simulation Parameters

Parameter	Value
Network size	500×500 m
System bandwidth	10 MHz
Transmit power of user nodes	10 dBm [42]
Edge delay (δ)	uniformly distributed in [0.125, 0.25]
Target transmission rate (Γ)	20 dB
Number of fog nodes (F)	30
Number of user nodes (U)	90
DNN association threshold (θ)	4
Mean task arrival rate (λ_u)	8 tasks/s
Processing density (c)	4 double-precision FLOPS/cycle [44]
Fog compute power (ν)	10^{10} cycle/s [43]
(Sub)task delay threshold (τ_a)	[5, 110] ms
CSR sparsity threshold ($\hat{\theta}$)	0.3 [31]

Proof: The matching μ is updated after swap matching at each iteration in Algorithm 2. Definition 3 implies that the utility of both the user and fog nodes improves after swap matching. Therefore, it suffices to show that the utility does not increase indefinitely with the swap matchings. The utility of both user and fog nodes depend on the total execution time T_{fa} according to Eq. (5) and Eq. (8). A swap matching increases the utility but also the load of fog nodes. The number of fog nodes in a network is limited, as the amount of computational resources they provide. Fog nodes can accept tasks for later execution by queuing them even when they are fully utilized; this increases the queuing time, thus, the total execution time as well. Consequently, the utility does not indefinitely increase, thus, the number of swap matchings as well as iterations is finite and DINA-O converges. ■

V. Performance Evaluation

This section evaluates the performance of DINA by extensive simulations based on a real dataset and several benchmarks.

A. Simulation Setup and Methodology

Experiments are carried out with a custom python network simulator built on top of the Caffe [40] deep learning framework. The Berkeley Deep Drive data set (BDD100k) is employed for both training and inference [41]. The dataset contains 120M images from 100K videos captured by cameras on self-driving cars; images are extracted from the videos every 10 seconds. Tasks are assumed to be independent from each other; they arrive at users according to a Poisson distribution with a mean of λ_u. Four well-known DNN models are employed as benchmarks: NiN and VGG16 as chain topologies as well as Alexnet and ResNet32 as DAG topologies (Fig. 3).

The deployment area of the network is a square grid of $500\,\text{m}^2$. The ratio between the number of fog nodes and user nodes is 1:3, according to [42, 43]. The bandwidth for user transmissions is 10 MHz, similar to that of 5G systems [42]. Computations are characterized by double precision floating point operations per cycle [44]. All the fog nodes have the same computing power. The delay thresholds are varied depending upon the size and type of the benchmark DNN. The parameters used in the simulation are reported in Table II.

For comparison purposes, three different schemes for partitioning and offloading are defined: random offloading of DNN inference tasks with DINA-P (Section III) as partition scheme (RANDP); random offloading of the DNN layers without partitioning (RAND); and a greedy algorithm that offloads the DNN layers to the nearest fog node without partitioning (GANC). DINA is also compared against the ECDI-L scheme in [7]. Unless otherwise stated, the data points in the results are the average of twenty replications for each experiment, with error bars representing the corresponding standard deviation.

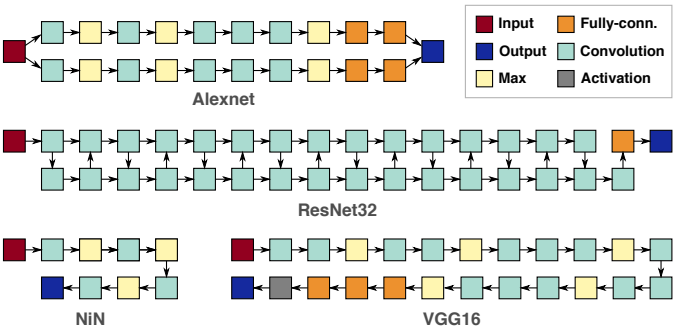


Fig. 3: Benchmark DNNs used in the evaluation, including both DAG (Alexnet and ResNet32) and chain (NiN and VGG16) topologies.

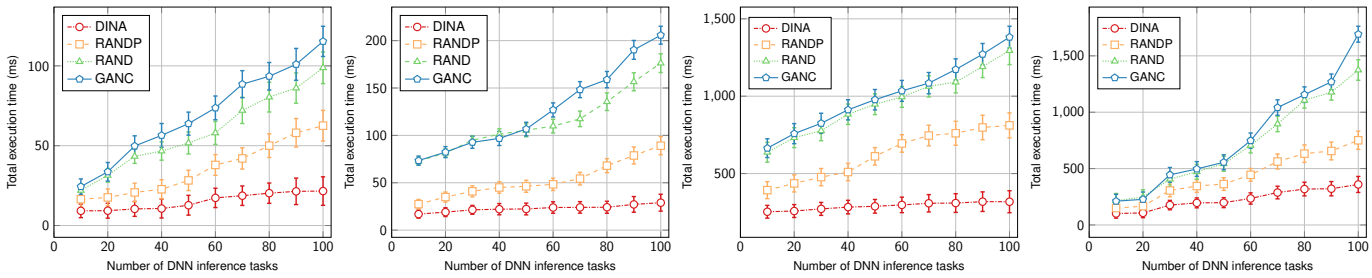
B. Obtained Results

Simulations results are presented next according to different metrics: total execution time, queuing time, and distribution of computation across fog nodes. Finally, DINA is compared against the considered schemes as well as the state of the art.

Total Execution Time. Fig. 4 illustrates the total execution time (T_{fa}) averaged over all DNN tasks for DINA and the other schemes (RANDP, RAND, GANC) for the considered DNN benchmarks (NiN, Alexnet, VGG16, ResNet32). DINA clearly achieves the lowest values in all cases. Moreover, it obtains the lowest increase in the total execution time with the number of tasks (always below 2.5 times), while the other schemes suffer from delays that grow substantially (roughly about one order of magnitude). This demonstrates the scalability of the proposed offloading scheme, together with the benefits in using DINA-P. Note that adaptive partitioning alone is generally beneficial; in fact, RANDP always performs better than other approaches with no partitioning (i.e., RAND and GANC). In particular, there is a higher gap when the DNN tasks increase over about 50 due to resource saturation.

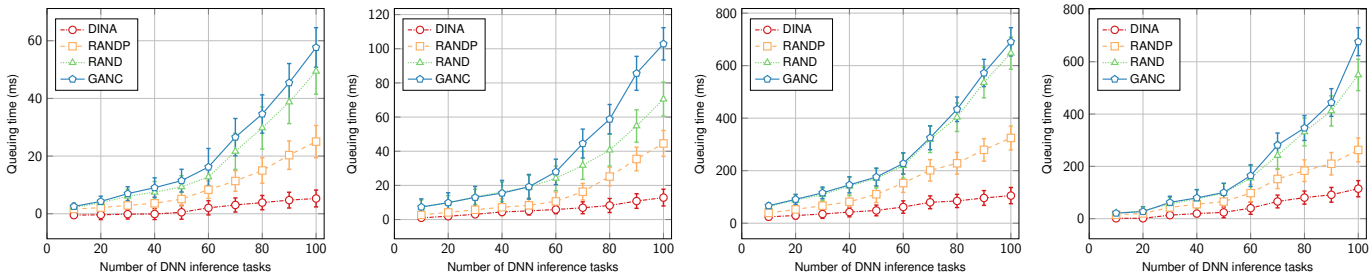
DINA obtains the best performance for the NiN benchmark (Fig. 4a). This is because NiN has a linear topology and also the smallest number of layers. The total execution time for Alexnet is higher (Fig. 4b), but still significantly lower than VGG16 (Fig. 4c) and ResNet32 (Fig. 4d). This happens as Alexnet has two independent paths that can be efficiently parallelized by all schemes; there is no major difference, instead, between the results for VGG16 and ResNet32, despite the different topologies (chain and DAG, respectively).

Queuing Time. Fig. 5 illustrates the average queuing time (T_{fa}^{que}) at fog nodes over all inference tasks for the considered schemes and the different DNN benchmarks. Also in this case DINA exhibits the best performance, corresponding to the



(a) (b) (c) (d)

Fig. 4: Total execution time of the different schemes as a function of the DNN tasks for (a) NiN, (b) Alexnet, (c) VGG16, and (d) ResNet32.



(a) (b) (c) (d)

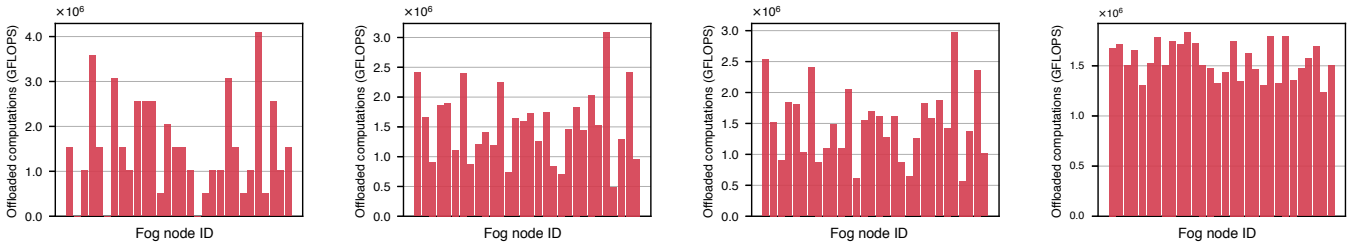
Fig. 5: Queuing time of the different schemes as a function of the DNN tasks for (a) NiN, (b) Alexnet, (c) VGG16, (d) ResNet32.

lowest queuing time, irrespective from the actual benchmark and the number of DNN tasks. In all cases, the queuing time of the different schemes for a small number of tasks is comparable, as fog nodes are lightly loaded. However, the obtained values rapidly increase for the schemes other than DINA, especially when the number of DNN tasks exceeds 50. This is consistent with the increase in the total execution time reported in Fig. 4, thereby revealing the prevalence of queuing time when fog nodes start being congested for GANC, RAND, and RANDP. This also explains why the trends for NiN (Fig. 5a), AlexNet (Fig. 5b), VGG16 (Fig. 5c), and ResNet32 (Fig. 5d) reflect those for the total execution time.

Distribution of Computation. Fig. 6 illustrates the amount of computation (in GFLOPS) of the individual fog nodes in the network when using the VGG16 benchmark. The figure shows how the most uneven distribution is obtained by GANC (Fig. 6a). This is expected, as GANC takes a greedy approach based on physical proximity. As a consequence, fog nodes that are close to many end devices quickly become overloaded; whereas those farther away may not be utilized, despite being available to execute offloaded (sub)tasks. Both RAND and RANDP result in a more even distribution (Figs. 6b–6c), due to their random selection policy. Note that their actual performance is rather different (as shown by Fig. 4), as the figures only provide the (average) total computation per fog node. Finally, DINA provides the most balanced utilization (Fig. 6d). Note that in this case DINA offloads 96% of the tasks, while 4% of them runs locally.

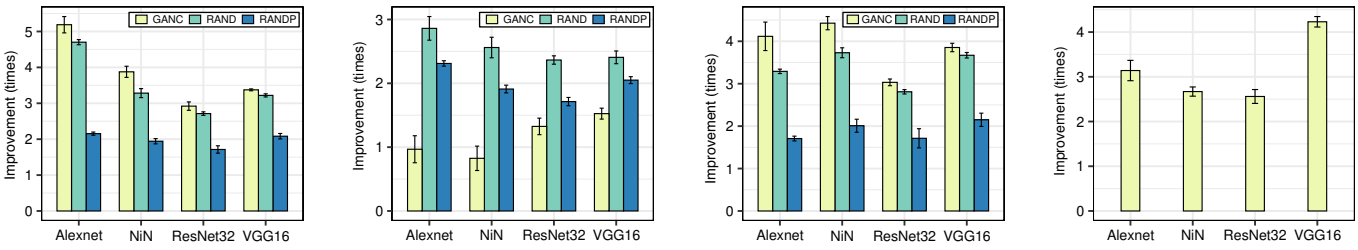
Improvement. Fig. 7 shows the improvement of DINA as the ratio between the time obtained with a certain scheme and that of DINA as a function of the considered DNN benchmarks. In particular, Fig. 7a shows the improvement in the total execution time. The figure clearly shows how DINA outperforms the other solutions, with improvements between 1.7 and 5.2. The best results are obtained for AlexNet, as it is the one that can be more easily parallelized. Fig. 7b shows the improvement in the transmission time. In this case, DINA achieves a performance that is 1.7 to 2.9 times better than RAND and RANDP. DINA obtains an improvement of about 1.5 over GANC for ResNet and VGG16; however, it actually performs worse than GANC for Alexnet and NiN in terms of transmission time. This happens as GANC greedily choses the nearest fog node, thereby maximizing the rate in Eq. (2) and reducing interference (as only one user is associated with a fog node). However, this is beneficial only when the parallel paths of the DNN are short (as in AlexNet and NiN), as the related intermediate processing and synchronization are not an issue. Fig. 7c shows the improvement in the queuing time. Here, DINA obtains the highest performance relative to GANC in this case (always over 3 times better), and an improvement between 1.7 and 3.8 over RAND and RANDP.

Finally, Fig. 7d shows the improvement of DINA against the state of the art, represented by ECDI-L in [7], in terms of total execution time. ECDI-L divides a DNN into two partitions by solving the minimum cut problem on the associated graph. The figure clearly shows how DINA achieves an improvement between 2.6 and 4.2. Such an improvement is higher for the benchmarks other than ResNet32 as these have less parallel paths; the longer the path, the better the performance of DINA. This happens as ECDI-L does not benefit much from obtaining the maximum cut of graphs with a relatively simple structure.



(a) (b) (c) (d)

Fig. 6: Amount of computations at individual fog nodes for the considered schemes: (a) GANC, (b) RAND, (c) RANDP, and (d) DINA-O.



(a) (b) (c) (d)

Fig. 7: Improvement of DINA against the other schemes in terms of: (a) total execution time, (b) transmission time, and (c) queuing time. (d) Improvement of DINA over ECDI-L [7] in terms of total execution time.

VI. Related Work

A few solutions in the literature have targeted DNN inference acceleration similar to this work. Among them, MoDNN [30] discusses a DNN partitioning and offloading scheme for mobile devices in a local cluster, connected through high-rate WiFi links. It partitions DNNs with a layer-aware scheme that leverages efficient representations, with the goal of reducing synchronization overhead. DINA adopts a similar partitioning approach, but operates on a tiered network, whose wireless links have significantly less bandwidth. Neurosurgeon [3] divides DNN computation between mobile devices and the cloud. It continuously monitors network conditions and offloads layers by minimizing latency or energy consumption, based on predictions from profiling data. DADS [7] splits a DNN into two partitions, one processed at the edge and the other at the cloud. It takes a graph-theoretic approach based on finding minimum cuts, which allows to precisely characterize DNNs described by a DAG. This work also considers a two-tier network but operates at a finer granularity than [3, 7], as it considers partitions smaller than a single layer. Moreover, the model here characterizes queuing that could occur at resource-constrained fog nodes, as opposed to the unlimited resources in the cloud. DDNN [16] is a framework for DNN computation across devices in three tiers: a local network, the edge, and the cloud. DNN inference takes place over stages (i.e., a set of consecutive layers) based on a certain prediction confidence. This requires a special training of the neural network; therefore, it cannot be used for pre-trained networks as those considered in this article. Lin et al. [17] also consider a three-tier network and DNN partitioned into stages. They propose algorithms to minimize latency and maximize throughput by considering each network tier as a whole. Instead, this work offloads DNN partitions to individual nodes, by explicitly keeping their resource utilization into account.

Other techniques for DNN inference acceleration include: pruning, which discards parameters from the source DNN [30]; model quantization, which restrict weights into discrete values [45]; and model compression, which removes redundancy [46]. However, these approaches reduce the accuracy of the DNN inference tasks; in contrast, this work leverages a pre-trained network as it is, without transforming the original model nor requiring re-training.

VII. Conclusion

This article introduced DINA, a distributed solution based on matching theory for joint partitioning and offloading of DNN inference tasks in fog networks. The objective was to reduce the total computation time while increasing the utilization of the resources in the network. Extensive simulations have demonstrated that DINA achieves a significant reduction in the total execution time due to fine-graining partitioning and effective adaptation to varying network conditions. In particular, the performance of DINA is up to 5 times better than other heuristics as well as 2.6 to 4.2 times faster than the state of the art. A promising future work is implementing DINA as a software framework for heterogeneous embedded devices. Another interesting research direction includes addressing distributed learning instead of DNN inference.

Acknowledgment

This work was partially supported by the Academy of Finland under grants number 299222 and 319710.

REFERENCES

[1] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ACM ASPLOS '17*. New York, NY, USA: ACM, 2017, pp. 615–629.

[4] Opensignal, "State of Mobile Networks: USA (January 2018)," https://www.opensignal.com/reports/2018/01/usa/state-of-the-mobile-network, online; Accessed July 31, 2019.

[5] OnePlus, "OnePlus 7 Pro Tech Specs," https://www.oneplus.com/7pro?/specs#/specs, online; Accessed July 31, 2019.

[6] NVIDIA Corporation, "Jetson Nano Developer Kit," https://developer.nvidia.com/embedded/jetson-nano-developer-kit, online; Accessed July 31, 2019.

[7] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, April 2019, pp. 1423–1431.

[8] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[9] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[10] H. Cao and J. Cai, "Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 752–764, Jan 2018.

[11] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys*, 4 2019.

[12] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[13] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

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

[15] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for Deep Neural Network," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 1396–1401.

[16] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *The 37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017)*, June 2017, pp. 328–339.

[17] C.-Y. Lin, T.-C. Wang, K.-C. Chen, B.-Y. Lee, and J.-J. Kuo, "Distributed deep neural network deployment for smart devices from the edge to the cloud," in *Proceedings of the ACM MobiHoc Workshop on Pervasive Systems in the IoT Era*, 2019, pp. 43–48.

[18] A. E. Roth and M. A. O. Sotomayor, *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*, ser. Econometric Society Monographs. Cambridge University Press, 1990.

[19] Z. Han, Y. Gu, and W. Saad, *Matching Theory for Wireless Networks*. Cham: Springer International Publishing, 2017.

[20] J. Zhao, Y. Liu, K. K. Chai, Y. Chen, and M. Elkashlan, "Many-to-many matching with externalities for device-to-device communications," *IEEE Wireless Communications Letters*, vol. 6, no. 1, pp. 138–141, Feb 2017.

[21] J. Zhao, Y. Liu, K. K. Chai, M. Elkashlan, and Y. Chen, "Matching with peer effects for context-aware resource allocation in D2D communications," *IEEE Comm. Lett.*, vol. 21, no. 4, pp. 837–840, April 2017.

[22] B. Di, L. Song, and Y. Li, "Sub-channel assignment, power allocation, and user scheduling for non-orthogonal multiple access networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 11, pp. 7686–7698, Nov 2016.

[23] Z. Qin and J. A. McCann, "Resource efficiency in low-power wide-area networks for IoT applications," in *The 2017 IEEE Global Communications Conference (GLOBECOM 2017)*, Dec 2017, pp. 1–7.

[24] "IEEE standard for adoption of OpenFog Reference architecture for fog computing," https://ieeexplore.ieee.org/servlet/opac?punumber=8423798, pp. 1–176, Aug 2018, IEEE Std 1934-2018.

[25] M. F. Hanif and P. J. Smith, "On the statistics of cognitive radio capacity in shadowing and fast fading environments," *IEEE Transactions on Wireless Communications*, vol. 9, no. 2, pp. 844–852, February 2010.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[27] A. Goldsmith, *Wireless Communications*. Cambridge University Press, Sep. 2005.

[28] S. Ko, K. Huang, S. Kim, and H. Chae, "Live prefetching for mobile computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 5, pp. 3057–3071, May 2017.

[29] A. Mukherjee, "Queue-aware dynamic on/off switching of small cells in dense heterogeneous networks," in *2013 IEEE Globecom Workshops (GC Wkshps)*, Dec 2013, pp. 182–187.

[30] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[31] R. Mehmood and J. Crowcroft, "Parallel iterative solution method for large sparse linear equation systems," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-650, Oct. 2005.

[32] T. Muhammed, R. Mehmood, A. Albeshri, and I. Katib, "SURAA: A novel method and tool for loadbalanced and coalesced SpMV computations on GPUs," *Applied Sciences*, vol. 9, no. 5, 2019.

[33] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[34] A. E. Roth and M. Sotomayor, "Two-sided matching," ser. Handbook of Game Theory with Economic Applications. Elsevier, 1992, vol. 1, ch. 16, pp. 485–541.

[35] F. Echenique and J. Oviedo, "A theory of stability in many-to-many matching markets," *Theoretical Economics*, vol. 1, no. 2, pp. 233–273, Jun. 2006.

[36] M. Pycia and M. B. Yenmez, "Matching with externalities," *Available at SSRN 2475468*, May 2015. [Online]. Available: https://ssrn.com/abstract=2475468

[37] K. Bando, R. Kawasaki, and S. Muto, "Two-sided matching with externalities: A survey," *Journal of the Operations Research Society of Japan*, vol. 59, no. 1, pp. 35–71, 2016.

[38] E. Bodine-Baron, C. Lee, A. Chong, B. Hassibi, and A. Wierman, "Peer effects and stability in matching markets," in *Algorithmic Game Theory*, ser. Lecture Notes in Comp. Sci., vol. 6982, 2011, pp. 117–129.

[39] A. E. Roth and J. H. V. Vate, "Random paths to stability in two-sided matching," *Econometrica*, vol. 58, no. 6, pp. 1475–1480, 1990.

[40] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[41] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "BDD100K: A diverse driving video database with scalable annotation tooling," *arXiv preprint arXiv:1805.04687*, 2018.

[42] M. S. Elbamby, M. Bennis, W. Saad, M. Latva-aho, and C. S. Hong, "Proactive edge computing in fog networks with latency and reliability guarantees," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, p. 209, Aug 2018.

[43] Y. Yu, J. Zhang, and K. B. Letaief, "Joint subcarrier and CPU time allocation for mobile edge computing," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

[44] R. Dolbeau, "Theoretical peak flops per instruction set: a tutorial," *The Journal of Supercomputing*, vol. 74, no. 3, pp. 1341–1377, Mar 2018.

[45] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 3123–3131.

[46] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," Published as a conference paper at ICLR 2016, 2015.