Xu, Yuzhe; Mohammed, Thaha; Francesco, Mario Di; Fischione, Carlo

Distributed Assignment with Load Balancing for DNN Inference at the Edge

Published in:
IEEE Internet of Things Journal

DOI:
10.1109/JIOT.2022.3205410

Published: 15/01/2023

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

Please cite the original version:
Xu, Y., Mohammed, T., Francesco, M. D., & Fischione, C. (2023). Distributed Assignment with Load Balancing for DNN Inference at the Edge. *IEEE Internet of Things Journal*, *10*(2), 1053-1065. Article 9882293. https://doi.org/10.1109/JIOT.2022.3205410

Distributed Assignment with Load Balancing for DNN Inference at the Edge

Yuzhe Xu, Thaha Mohammed, Mario Di Francesco, and Carlo Fischione

***Abstract*—Inference carried out on pre-trained deep neural networks (DNNs) is particularly effective as it does not require re-training and entails no loss in accuracy. Unfortunately, resource-constrained devices such as those in the Internet of Things may need to offload the related computation to more powerful servers, particularly, at the network edge. However, edge servers have limited resources compared to those in the cloud; therefore, inference offloading generally requires dividing the original DNN into different pieces that are then assigned to multiple edge servers. Related approaches in the state of the art either make strong assumptions on the system model or fail to provide strict performance guarantees. This article specifically addresses these limitations by applying distributed assignment to deep neural network inference at the edge. In particular, it devises a detailed model of DNN-based inference, suitable for realistic scenarios involving edge computing. Optimal inference offloading with load balancing is also defined as a multiple assignment problem that maximizes proportional fairness. Moreover, a distributed algorithm for DNN inference offloading is introduced to solve such a problem in polynomial time with strong optimality guarantees. Finally, extensive simulations employing different datasets and DNN architectures establish that the proposed solution significantly improves upon the state of the art in terms of inference time (1.14 to 2.62 times faster), load balance (with a Jain's fairness index of 0.9), and convergence (one order of magnitude less iterations).**

***Index Terms*—Distributed inference, DNN offloading, assignment problems, edge computing**

I. Introduction

Machine learning and artificial intelligence – particularly, deep learning – are being more and more applied to diverse scenarios [1]. In many cases of practical importance, learning involves deep neural networks (DNNs) running at resource-constrained embedded devices such as mobile phones, wearables, and smart objects in next-generation networks [2, 3]. Unfortunately, the related algorithms generally require a significant amount of computational resources to achieve satisfactory results. In this respect, several techniques have been proposed to specifically address the limitations of resource-constrained devices, including hardware-accelerated computing and model simplification (i.e., pruning, quantization, and compression) for DNN inference tasks [4, 5] as well as federated learning for DNN training tasks [6–9]. However, the above-mentioned approaches for DNN inference cannot always be applied or might reduce accuracy, especially when the device capabilities are extremely limited, such as in the Internet of Things (IoT) [10]. In contrast, DNN training with federated learning incurs a significant overhead, as individual devices have to exchange data with a coordinator to gain global knowledge [11–13].

Computation offloading is a general approach to overcome the constraints of embedded devices through resources offered by third-party services over the Internet [14, 15]. Such an approach can as well be applied to DNN computation: devices do not need to perform any training but rather transfer data to one or more powerful servers that then take care of the actual processing, or just carry out inference on pre-trained DNNs [16–18]. The latter option is particularly effective as it does not need re-training and entails no loss in accuracy [16, 17, 19]. Offloading had originally leveraged the cloud computing paradigm [20], while it has more recently shifted towards edge computing as a compelling alternative [21, 22]. Indeed, the edge is an infrastructure of server-class devices that are close to end users and can be reached with a low latency [23]. The main challenge here is that the resources provided by edge servers – in terms of both computation and bandwidth – are limited compared to those in the cloud. As a consequence, they may not be enough to run inference on the original DNN as a whole, especially if the trained model is very large [14].

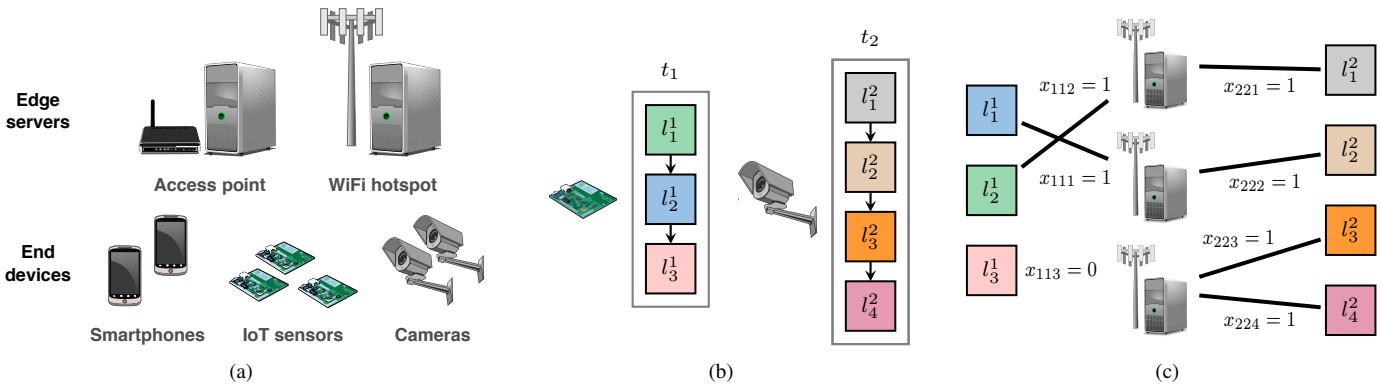
To overcome this issue, the original DNN is divided into different pieces – generally corresponding to the individual layers therein – which are then assigned to individual edge servers [16]. Both centralized and distributed approaches have been proposed accordingly, for instance, based on graph theory [16, 24], matching theory [17], and convex optimization [18]. However, solutions in the state of the art suffer from important limitations. In fact, most schemes built on solid theoretical foundations are centralized and make strong assumptions on the system model, such as that DNNs can only be divided into two pieces [16]. Other approaches are distributed and able to handle the complexities of real settings [17, 18], but they are rather involved and fail to provide strict performance guarantees, especially in terms of efficient and fair use of resources (Section V).

This article explicitly addresses these limitations by applying distributed assignment to DNN inference at the edge. Specifically, the offloading process is modeled as an assignment problem [25], wherein inference tasks are associated to edge servers while balancing the resulting load. In particular, a distributed assignment mechanism is devised to maximize proportional fairness, a widely used metric in network systems which jointly characterizes resource utilization and load bal-

This work was partially supported by the Academy of Finland under grants number 326346 and 332307, the Digital Futures research project DEMOCRITUS, and the VR project MALEN.

Y. Xu and C. Fischione are with the School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden (e-mail: {yuzhe,carlofi}@kth.se). T. Mohammed and M. Di Francesco are with the Department of Computer Science, Aalto University, Espoo, Finland (e-mail: {thaha.mohammed,mario.di.francesco}@aalto.fi).

 DOI: 10.1109/JIOT.2022.3205410



(a) (b) (c)

Fig. 1: DNN inference offloading at the edge. (a) Reference architecture with heterogeneous end devices and more powerful edge servers. (b) Sample DNN inference tasks t_1 and t_2 associated with different devices. Each task relies on a certain DNN architecture with multiple layers l_i. (c) Sample assignment between these tasks and three edge servers.

ancing as a sum of logarithmic utility functions [26, 27]. The proposed approach is expressive, analytically tractable, and able to achieve a near-optimal solution in terms of resource utilization and inference latency. To the best of the authors' knowledge, this is the first work to provide strong optimality guarantees in offloading of DNN inference with a distributed scheme.

In detail, the main contributions of this work are the following.

- It devises **a detailed model of the DNN inference process**, suitable for realistic scenarios involving edge computing. It also defines optimal inference offloading with load balancing as a multiple assignment problem that maximizes proportional fairness (Section II).
- It introduces **a distributed algorithm that solves the multiple assignment problem** in polynomial time and achieves strong optimality guarantees (Section III).
- Extensive simulations employing different datasets and DNN architectures (Section IV) establish that the proposed solution **significantly improves upon the state of the art** in terms of inference time (1.14 to 2.62 times faster), load balance (with a Jain's fairness index of 0.9), and convergence (one order of magnitude less iterations).

II. DNN Inference at the Edge

This section introduces the system model[1] of an edge computing scenario wherein DNN inference tasks are offloaded from end devices to edge servers. It first characterizes the network and the inference tasks, then discusses the computation aspects related to the DNN inference process. It finally formulates an optimization problem for offloading DNN inference tasks with load balancing. The notation used in the article is summarized in Table I.

A. Network and Inference Tasks

The reference network architecture includes two different components [28] distributed over a certain geographical area (Fig. 1a): a set $\mathcal{S}$ of $n = |\mathcal{S}|$ edge nodes (i.e., servers) and a set $\mathcal{D}$ of $D = |\mathcal{D}|$ end devices. Edge servers are connected to each other and to the cloud through high-speed dedicated links [29]; as a consequence, the time needed by edge servers to exchange intermediate results between each other is considered negligible. In contrast, end devices exchange data with edge servers in their communication range over a shared wireless channel (i.e., through WiFi) with a total bandwidth of B, according to [16].

End devices execute DNN inference tasks indicated as $\mathcal{T} = \{t_1, t_2, \cdots, t_T\}$. Specifically, a given task $t \in \mathcal{T}$ evaluates (i.e., obtains the output of) a DNN, represented as a linear graph $\mathcal{G}^t = (\mathcal{E}^t, \mathcal{L}^t)$ whose nodes in $\mathcal{L}^t = \{l_1^t, l_2^t, \cdots, l_n^t\}$, i.e., l_i^t are the layers of the corresponding DNN. Here, l_1^t and l_n^t indicate the input layer and the output layers, respectively; moreover, edges $(l_i^t, l_{i+1}^t) \in \mathcal{E}^t$ imply that l_{i+1}^t depends on l_i^t, therefore, it must be evaluated first (Fig. 1b). For conciseness, the index t is dropped from the notation in the rest of the article; the related discussion refers to a given task, without loss of generality.

B. DNN Inference Process

A given layer is either executed locally at the end devices or sent to edge servers (with higher computational power) together with its input. Each edge server k has a constant CPU-cycle frequency of ν_k [30] and executes a layer immediately if enough computational resources are available. Otherwise, the edge server adds the layer to a local FIFO execution queue with size Q_k proportional to its computing power [17].

The input and output size of layer j at end device i are denoted as I_{ij}^s and O_{ij}^s, respectively. The number of cycles to process one element of the input (i.e., the task density) is indicated as c. DNN layers are described by matrices that either denote the corresponding weights or the feature map, depending on their type. The total number of floating point operations for a single convolutional layer j (assuming an implementation based on a sliding window) from device i is given by

$$C_{ij} = 2 \cdot I_{ij}^h \cdot I_{ij}^w \cdot (c_{in} \cdot K_{ij}^w \cdot K_{ij}^h + 1) \cdot c_{out}$$

where: I_{ij}^h, I_{ij}^w, K_{ij}^w, and K_{ij}^h are the height and width of the input feature map and the kernel, respectively; c_{in} and c_{out}

[1]The rest of the discussion makes certain simplifying assumptions on the system model for the sake of clarity and analytical tractability. Additional considerations on these assumptions will be provided in Section IV-B.

TABLE I
SUMMARY OF KEY NOTATION

Symbol	Description
$\mathcal{S}$, k	Set of edge nodes (servers) and a given server
$\mathcal{D}$, i	Set of end devices and a given end device
$\mathcal{G}(\mathcal{E},\mathcal{L})$	DNN graph with layers $\mathcal{L}$ and edges $\mathcal{E}$ for task t
j	A given layer in $\mathcal{L}$
m	Cardinality of layers $\mathcal{L}$
n	Cardinality of edge nodes
ν_k	Computing power of edge node k
Q_k	Queue length of edge node k
C_{ij}	FLOPs for a convolutional layer j from device i
F_{ij}	FLOPs for a fully-connected layer j from device i
T^c_{ijk}	Time for computing a convolutional layer j at server k
T^f_{ijk}	Time for computing a fully-connected layer j at server k
I^s_{ij}	Size of the input layer j at end device i in bits
O^s_{ij}	Size of the output layer j at end device i in bits
T_{ijk}	Total time for layer j, from user i at edge node k
T^e_{ijk}	Execution time for layer j, from user i at edge node k
T^t_{ijk}	Transmission time for layer j, from user i at edge node k
T^q_{ijk}	Queuing time for layer j, from user i at edge node k
B	Total bandwidth of shared wireless channel
$\mathcal{X}$	Set of final assignments (j,k) mapping layer j to server k
$\mathcal{L}(k)$	Subset of layers ($\subseteq \mathcal{L}$) that can use server k
$\mathcal{S}(j)$	Subset of servers ($\subseteq \mathcal{S}$) that can run layer j
$\mathcal{Y}$	Set of feasible assignments (j,k)
x_{ijk}	Binary variable denoting assignment of layer j to node k
τ_{ijk}	Time threshold for layer j from user i at server k
n^+_k, n^-_k	Upper and lower bound on layers assigned to server k
c	Task processing density of edge nodes
a_{jk}	Generic utility for executing a task j at server k
Y	Cardinality of the set of feasible assignments $\mathcal{Y}$
$\boldsymbol{p}$	Price of a server to run the layers
$\boldsymbol{\pi}$	Profits associated with layers as an incentive to servers

are the number of channels in the input feature maps and the output. Thus, the execution time for a single convolutional layer j from device i at edge server k is

$$T^c_{ijk} = (C_{ij} \cdot c)/\nu_k.$$

Similarly, the number of floating point operations for a (fully-connected) feed-forward layer j from device i is

$$F_{ij} = (I_{ij} \cdot O_{ij} + O_{ij} \cdot (I_{ij} - 1)) = (2I_{ij} - 1)O_{ij}$$

where I_{ij} and O_{ij} are the input and output dimensions. The related computation time at edge server k is

$$T^f_{ijk} = (F_{ij} \cdot c)/\nu_k.$$

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^e_{ijk} for executing a deep learning layer j from device i is $T^e_{ijk} = T^c_{ijk}$ for a convolutional layer and $T^e_{ijk} = T^f_{ijk}$ for a fully-connected layer.

Next, the time needed to exchange inputs and outputs between an end device and an edge server are derived. In particular, the time taken by an end device i to send the DNN partition j to edge server k and receive the intermediate output back (where applicable[2]) is

$$T^t_{ijk} = (I^s_{ij} + O^s_{ij})/B$$

where O^s_{ij}=0 if the layer j+1 is also executed at server k.

Recall that offloaded layers may not be executed immediately at the edge server, hence, they may be subject to a queuing time

$$T^q_{ijk} = \sum_{j_q \in Q_k, j \neq j_q} \frac{\text{flops}(j_q) \cdot c}{\nu_k}$$

where $\text{flops}(j_q)$ are the floating point operations for layer j_q in the execution queue $q \in Q_k$ before task j at edge server k.

Then the total execution time for a layer j offloaded by end device i to an edge server k is:

$$T_{ijk} = T^t_{ijk} + T^e_{ijk} + T^q_{ijk} \tag{1}$$

where: T^t_{ijk} is the time for transmitting the input of layer j to server k and the possible intermediate output from server k back to the device; T^e_{ijk} is the time taken for computing layer j at server k; and T^q_{ijk} is the waiting time of layer j at k.

C. The DNN Inference Offloading Problem

Now that the system model has been introduced, it remains to characterize the actual offloading process. Recall that an end device can offload the execution of individual layers of a given DNN inference task to edge servers. Edge servers can run these layers faster than end devices; however, they may not be able to process requests immediately due to precedence constraints and the load coming from several nodes in the network. Moreover, running layers at edge servers requires exchanging the input and output of individual layers with end devices over a bandwidth-constrained channel. Therefore, offloading DNN inference must explicitly these factors into account and reduce the total inference time while balancing the load of the edge servers in the network.

For this reason, the offloading process is modeled as an assignment problem: individual layers of DNN inference tasks requested at end devices are mapped onto edge servers for execution (Fig. 1c). Layer j executed by server k is indicated as the (j,k) pair in the final assignment $\mathcal{X}$. Not every layer can be assigned to any server and vice versa. Specifically, $\mathcal{L}(k) \subseteq \mathcal{L}$ is the set of layers that can use server k, whereas $\mathcal{S}(j) \subseteq \mathcal{S}$ is the set of servers that can run layer j. Furthermore, $\mathcal{Y}$ is a feasible assignment[3] as the set of all pairs such that $(j,k) \in \mathcal{Y}$ if and only if $k \in \mathcal{S}(j)$ as well as $j \in \mathcal{L}(k)$.

The assignment problem describing the offloading process is formally defined next.

[2]In particular, intermediate output does not need to be transmitted for consecutive layers that are all executed at the same device.

[3]For now, it suffices to assume that such an assignment exists; additional considerations on feasibility will be given in the next section.

Problem 1 (DNN Inference Offloading with Load Balancing)**.** The problem of offloading DNN inference with load balancing is formulated as follows:

$$\max_{\boldsymbol{x}} \quad -\sum_{i\in\mathcal{D}}\sum_{(j,k)\in\mathcal{Y}} \log(T_{ijk})\cdot x_{ijk} \tag{2a}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{S}(j)} x_{ijk} \leq 1, \quad \forall j\in\mathcal{L}, i\in\mathcal{D} \tag{2b}$$

$$T_{ijk} \leq \tau_{ijk}, \quad \forall (j,k)\in\mathcal{X}, i\in\mathcal{D} \tag{2c}$$

$$n_k^- \leq \sum_{j\in\mathcal{L}(k), i\in\mathcal{D}} x_{ijk} \leq n_k^+, \quad \forall k\in\mathcal{S}, \tag{2d}$$

$$x_{ijk}\in\{0,1\}, \quad \forall (j,k)\in\mathcal{Y}, i\in\mathcal{D} \tag{2e}$$

In detail, Eq. (2a) is the objective function, which expresses load balancing in terms of proportional fairness by maximizing the sum of the logarithms for a certain utility [26]. Here, the utility is inversely proportional to the total inference time T_{ijk} [as in Eq. (1)], therefore explaining the negative sign before the summation. Clearly, the total inference time is considered for the layers of all DNN inference tasks offloaded in a feasible assignment $\mathcal{Y}$. The decision variables $x_{ijk}\in\{0,1\}$ express whether layer j from device i is offloaded to edge server k (i.e., $x_{ijk}=1$) or not (i.e., $x_{ijk}=0$). The optimization is subject to several constraints as follows. Eq. (2b) indicates that not all edge servers may be assigned layers. Eq. (2c) signifies that layer j is only offloaded if it is beneficial, i.e., if its total execution time – including the computation at the edge server in addition to the time for transferring intermediate input / output data – is lower than time τ_{ijk} for local execution at the device. Eq. (2d) expresses an upper and a lower bound (i.e., n_k^+ and n_k^-, respectively) on the number of layers assigned to a given server k. Finally, Eq. (2e) states that decision variables are binary.

Problem 2 is an instance of a non-linear binary integer programming model, thus, it is NP-hard [31]. The rest of the article targets finding an approximate yet near-optimal solution to the problem. In particularly, it focuses on a distributed solution, as it is more suitable for the scenario with multiple edge servers and possibly a large number of end devices.

III. Distributed Assignment for DNN Inference

This section introduces and analytically characterizes a distributed algorithm that obtains a near-optimal solution to the problem introduced above. For the sake of exposition, the rest of the discussion addresses first a *multiple* assignment problem – wherein more than one tasks can be assigned to a single server – with a linear objective function. It then extends the proposed algorithm to the case of a logarithmic objective function expressing proportional fairness. The rest of the section focuses on offloading inference tasks involving a single device[4] for clarity.

[4]Accordingly, the index i denoting a specific device is dropped from all notation for conciseness.

A. Multiple Assignment

The multiple assignment problem allows more than one layer to be assigned to the same server as follows.

Problem 2 (Multiple Assignment)**.** The multiple assignment problem is defined as:

$$\max_{\boldsymbol{x}} \quad \sum_{(j,k)\in\mathcal{Y}} a_{jk}x_{jk} \tag{3a}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{S}(j)} x_{jk} = 1\,, \quad \forall j\in\mathcal{L}\,, \tag{3b}$$

$$n_k^- \leq \sum_{j\in\mathcal{L}(k)} x_{jk} \leq n_k^+\,, \quad \forall k\in\mathcal{S}\,, \tag{3c}$$

$$x_{jk}\in\{0,1\}\,, \quad \forall (j,k)\in\mathcal{Y}\,, \tag{3d}$$

where a_{jk} expresses a generic[5] utility, $|\mathcal{L}|\geq|\mathcal{S}|$ and $|\mathcal{S}|=n$. Note the linear objective function in Eq. (3a); here, the constraints state that: all layers are assigned [Eq. (3b)], those running at edge servers are within given lower / upper bounds [Eq. (3c)], and decision variables are binary [Eq. (3d)].

The problem is addressed by transforming it into an equivalent formulation in two steps. First, the problem is expressed as a symmetric assignment by replacing each original server by a *virtual server* that processes at most one layer. Accordingly, $\mathcal{S}$ is replaced by $\mathcal{S}'=\mathcal{S}'_+\cup\mathcal{S}'_-$, wherein $\mathcal{S}'_+$ and $\mathcal{S}'_-$ are the sets of $n_k^+-n_k^-$ and n_k^- virtual servers, respectively. Second, a supersource s is added to the network and is connected to each virtual server $k\in\mathcal{S}'_+$ with zero cost. This allows to express the problem as a minimum-cost flow:

$$\min_{\boldsymbol{x}} \quad \sum_{(j,k)\in\mathcal{Y}} -a_{jk}x_{jk} \tag{4a}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{S}'(i)} x_{jk} = 1\,, \quad \forall i\in\mathcal{L}\,, \tag{4b}$$

$$\sum_{j\in\mathcal{L}(k)} x_{jk} = 1\,, \quad \forall k\in\mathcal{S}'_-\,, \tag{4c}$$

$$\sum_{j\in\mathcal{L}(k)} x_{jk} + x_{sk} = 1\,, \quad \forall k\in\mathcal{S}'_+\,, \tag{4d}$$

$$\sum_{k\in\mathcal{S}'_+} x_{sk} = m^+ - m\,, \tag{4e}$$

$$0\leq x_{jk}\,, \quad \forall (j,k)\in\mathcal{Y}\,, \tag{4f}$$

The meaning of the constraints is the following. Eq. (4b) and Eq. (4c) indicate that the flow supply of each layer / server is one unit, while Eq. (4d) states that a unitary flow eventually reaches each virtual server. Moreover, Eq. (4e) signifies that s is the supersource and that the flow it generates is m^+-m units, with $m^+=\sum_{k\in\mathcal{S}} n_k^+$. Finally, Eq. (4f) enforces non-negative flows.

It is now possible to consider the reformulated problem according to the duality theory [31]. To this end, the Lagrangian function is first derived by combining the objective function

[5]The rest of the discussion uses such a utility according to notation commonly used in the literature [25], also because the analytical characterization derived next applies to multiple assignment in general. In practice, the utility indicates the total inference time in Eq. (1) for the problem considered here.

with the constraints through the Lagrangian multipliers. For clarity, the multipliers are divided based on the elements they refer to: π_j for the layers [Eq. (4b)]; p_k for the servers [Eq. (4c) and (4d)], and λ for the supersource [Eq. (4e)]. Accordingly, the Lagrangian function is:

$$\begin{aligned}&\mathcal{L}(\boldsymbol{x},\boldsymbol{\pi},\boldsymbol{p},\lambda)\\&=\sum_{(j,k)\in\mathcal{Y}}(-a_{jk}+\pi_j+p_k)x_{jk}+\sum_{k\in\mathcal{S}'_+}(p_k-\lambda)x_{sk}\\&\quad-\sum_{j\in\mathcal{L}}\pi_j-\sum_{k\in\mathcal{S}'}p_k-\lambda(m-m^+),\end{aligned}\tag{5}$$

The corresponding dual problem can then be written as:

$$\min_{\boldsymbol{\pi},\boldsymbol{p}}\quad\sum_{j\in\mathcal{L}}\pi_j+\sum_{k\in\mathcal{S}'}p_k-\lambda(m^+-m)\tag{6a}$$

$$\text{s.t.}\quad\pi_j+p_k\geq a_{jk}\,,\quad\forall(j,k)\in\mathcal{Y}\,,\tag{6b}$$

$$\lambda\leq p_k\,,\quad\forall k\in\mathcal{S}'_+\,,\tag{6c}$$

The duality theory establishes a relationship between the (original) primal problem and the dual problem. Such a relationship is characterized in terms of the so-called *complementary slackness* (CS) condition, which describes how the constraints in the two problems are binding [31]. For the multiple assignment problem, the following variant of CS is employed.

Definition 1 [ϵ-complementary slackness (ϵ-CS)]**.** Given a positive scalar ϵ, an assignment $\mathcal{X}=\{(j,k)|j=1,\cdots,m\}$ and a pair $(\boldsymbol{\pi},\boldsymbol{p})$ satisfy the ϵ-CS condition if:

$$\pi_j+p_k\geq a_{jk}-\epsilon\,,\qquad\forall(j,k)\in\mathcal{Y}\,,\tag{7}$$

$$\pi_j+p_k=a_{jk}\,,\qquad\forall(j,k)\in\mathcal{X}\,,\tag{8}$$

$$p_k\leq\min_{l\in\mathcal{S}'_+,\exists(j,l)\in\mathcal{X}}p_l\,,\qquad\forall k\in\mathcal{S}'_+,\nexists(j,k)\in\mathcal{X}\,,\tag{9}$$

Note that the ϵ-CS condition above is expressed in terms of the dual problem through the Lagrangian function in Eq. (5). Such a condition allows to characterize the optimality of a solution to the multiple assignment problem as follows.

Proposition 1 (Proposition 7.7 in [25])**.** *Let $\mathcal{X}$ be a feasible assignment for the problem together with a dual variable pair $(\boldsymbol{\pi},\boldsymbol{p})$ satisfying ϵ-CS conditions. Then $\mathcal{X}$ is within $m\epsilon$ of the optimal solution to Problem 2.*

B. A Distributed Algorithm for the Multiple Assignment Problem

The previous discussion has introduced a theoretical framework to characterize a solution for the multiple assignment in Problem 2. Next, a distributed algorithm to find a near-optimal solution is devised accordingly. The main idea is to consider the assignment problem as in an economic system: finding an optimal assignment corresponds to reaching an economic[6] equilibrium. In this context, the pair $(\boldsymbol{\pi},\boldsymbol{p})$ offers an economic interpretation: $\boldsymbol{\pi}$ are profits associated with layers, as an incentive for servers to run them; whereas $\boldsymbol{p}$ is the price of a server, as the cost to run layers. These two quantities are then leveraged to design an auction mechanism that enforces the ϵ-CS condition in Proposition 1 to obtain a near-optimal solution to Problem 2.

[6]A similar analogy is employed in [25], which presents *auction algorithms* to solve different types of assignment problems. This section uses the same terminology only for clarity, whereas an analysis of economic properties of the proposed algorithm is beyond the scope of this work.

Algorithm 1: DAMA executed at end device i for each layer j in a DNN inference task

1 Initialize $\pi_j=\varnothing$, $k_j=\varnothing$, and $\boldsymbol{p}^j$
2 **if** $k_j=\varnothing$
3 $k'_j\leftarrow\arg\max_{k\in\mathcal{S}}\{a_{jk}-p^j_k\}$
4 $u_j\leftarrow\max_{k\in\mathcal{S}}\{a_{jk}-p^j_k\}$
5 $\omega_j\leftarrow\max_{k\in\mathcal{S}\setminus\{k'_j\}}\{a_{jk}-p^j_k\}$
6 $b_j\leftarrow p^j_{k'_j}+u_j-\omega_j+\epsilon$
7 place bid b_j for server k'_j
8 listen for response, ACK / NACK and $p_{k'_j}$, $p^j_{k'_j}\leftarrow p_{k'_j}$
9 **if** ACK
10 offload layer to server k'_j, $\pi_j\leftarrow a_{jk'_j}-p_{k'_j}$, $k_j\leftarrow k'_j$
11 **if** receive request from server k with bid b_k
12 **if** $b_k-\pi_j\geq\epsilon$
13 $\pi_j\leftarrow b_k$, send ACK and π_j to server k
14 send NACK and π_j to server k_j
15 offload layer to server k, $k_j\leftarrow k$
16 **else** send NACK and π_j to server k

The proposed algorithm involves two distinct phases: a forward auction, wherein devices place bids for edge servers to run their layers; and a reverse auction, wherein edge servers place bids for layers. Such an algorithm – called *distributed auction for multiple assignment* (DAMA) – is fully distributed, as it runs at both end devices and edge servers. It also operates online, since prices are updated asynchronously as bids arrive. Moreover, the algorithm only leverages local information, as servers (end devices) have their own evaluation of profits (prices). These local values, not necessarily corresponding to the correct ones, are indicated as the vectors $\boldsymbol{p}^j=\{p_k\}$ for server $\{k=1,\ldots,m\}$ at end device i for each layer j as well as $\boldsymbol{\pi}^k=\{\pi_j\}$ for layer $\{j=1,\ldots,n\}$ from user i at server k. All entities running the algorithm share the same value of ϵ, decided beforehand.

Algorithm 1 describes the operations of DAMA as executed at end device i for each layer j of a DNN inference task. The algorithm runs each time a new DNN inference task arrives at the device, and concerns all the DNN layers therein. Initially, the device obtains the benefit a_{jk} and the price p_k of each edge server k. At first, no server is assigned to the layer, so the device makes a bid for the the server offering the highest gain (lines 3–6). Specifically, the device identifies the server k'_i that provides the largest difference between the layer's benefit and the server price (line 3). It also obtains the actual value u_j corresponding to such a gain (line 4) as well as the second-best value ω_j (line 5). These values are then employed to derive the bid b_j (line 6), which is sent to server k'_j (line 7). Upon receiving a response from the server, the device updates the local price with the up-to-date value (line 8). If the response also contains a positive acknowledgment (ACK), layer j is associated with server k'_j (lines 9–10). Afterwards, the device listens for bids sent by the edge servers (line 11). If a new bid b_k arrives from server k, it is accepted only if the difference

Algorithm 2: DAMA executed at server k

1 Initialize $p_k = 0$, $\boldsymbol{\jmath}_k = \varnothing$, $\boldsymbol{\pi}^k = 0$
2 **if** receive request for layer j in user i with bid b_j
3 **if** the cardinality $|\boldsymbol{\jmath}_k| < n_k^+$
4 $\pi_j^k \leftarrow a_{jk} - b_j$, $\boldsymbol{\jmath}_k \leftarrow \boldsymbol{\jmath}_k \cup \{j\}$
5 **if** $|\boldsymbol{\jmath}_k| = n_k^+$
6 $p_k \leftarrow \min_{l \in \boldsymbol{\jmath}_k} a_{lk} - [\boldsymbol{\pi}^k]_l$
7 send ACK and p_k to device i accepting layer j
8 **else if** $b_j \geq p_k + \epsilon$
9 $j'_k \leftarrow \arg\min_{l \in \boldsymbol{\jmath}_k} a_{lk} - \pi_l^k$
10 $\boldsymbol{\jmath}_k \leftarrow \boldsymbol{\jmath}_k \setminus \{j'_k\} \cup \{j\}$
11 $\pi_j^k \leftarrow a_{jk} - b_j$
12 $p_k \leftarrow \min_{l \in \boldsymbol{\jmath}_k} a_{lk} - \pi_l^k$
13 send ACK and p_k to device i accepting layer j
14 send NACK and p_k to device that requested layer j'_k
15 **if** the cardinality $|\boldsymbol{\jmath}_k| < n_k^-$ **and** all layers are assigned
16 $j_k \leftarrow \arg\max_{j \in \mathcal{L}} \{a_{jk} - \pi_j^k\}$
17 $u_k \leftarrow \max_{j \in \mathcal{L}} \{a_{jk} - \pi_j^k\}$
18 $\omega_k \leftarrow \max_{j \in \mathcal{L} \setminus \{j_k\}} \{a_{jk} - \pi_j^k\}$
19 $b_k \leftarrow \pi_{j_k}^k + u_k - \omega_k + \epsilon$
20 place bid b_j for layer j_k at user i
21 listen for response with ACK / NACK and π_{j_k}, $\pi_{j_k}^k \leftarrow \pi_{j_k}$
22 **if** ACK
23 accept layer j_k, $j_k \leftarrow \boldsymbol{\jmath}_k \cup \{j_k\}$

between the bid and the profit is at least equal to ϵ (line 12). In that case, the device updates the layer's profit, sends it to server k with an ACK, and to the previously-selected server k_j with a negative acknowledgment (NACK, lines 13–15). Otherwise, it sends the profit to server k with a NACK (line 16).

Algorithm 2 describes the operations of DAMA as executed at each server k. After initializing prices and profits to zero, the server listens for bids from devices (lines 1–14). When a bid arrives, the server accepts the n_k^+ best layers, i.e., those with the highest bids. In such a case, the server updates the profit of each layer and its own price; then, it sends the updated price to the device with an ACK (lines 3–7). If the server has already taken n_k^+ layers for execution, it still accepts layers but only if the difference between the bid and the price is at least equal to ϵ (line 8). If so, the server updates its price, sends it to the device requesting to offload j with an ACK, and to the device of the previously-selected layer j'_k with a NACK (lines 9–13). Otherwise, it still sends the price to device i but with a NACK (line 14). Bids are then to devices if all layers are assigned and the server has accepted less than n_k^- layers (lines 15–23) similar to the first part of Algorithm 1. First, the server identifies the layer j_k that provides the largest gain, obtains the value for that and for the second-best gain, then derives the bid (lines 16–19). It finally places the bid, updates the local profit with the up-to-date value and possibly the association if the response contains an ACK (lines 20–23).

The following characterizes the time complexity of DAMA.

Proposition 2. *Assume that a feasible assignment $\mathcal{Y}$ exists for Problem 2, with $Y = |\mathcal{Y}|$, and that $\epsilon > 0$ is given. The time complexity of DAMA is $\mathcal{O}(mY\lceil\Delta/\epsilon\rceil)$, where $m = |\mathcal{L}|$ and $\Delta = \max_{(j,k)\in\mathcal{Y}} a_{jk} - \min_{(j,k)\in\mathcal{Y}} a_{jk}$.*

Proof. DAMA is shown to terminate with a feasible assignment first, since such an assignment exists by hypothesis. Recall that DAMA consists of a forward and a reverse auction, executed sequentially. Both types of auctions work similarly, with roles and profits / prices exchanged; therefore, it is enough to show that the forward action terminates. This can be proven by contradiction as follows. Assume that DAMA does not terminate; this implies that some end device placed infinite bids for a certain layer. Since each bid increases the price of a server by at least ϵ, the price of these servers would go to infinity, while the difference between the benefits and the prices would reach minus infinity. As a consequence, at least one layer remained unassigned in a finite amount of time, which contradicts the existence of a feasible assignment.

Given that DAMA terminates, its time complexity can be described based on two factors: the maximum number of price increments and the maximum number of requests in between them. First, the maximum number of price increments occurs when end devices place bids with the minimum allowed increase of ϵ. In particular, such a number of increments is equal to $\lceil\Delta/\epsilon\rceil$, where Δ denotes the maximum variation in price corresponding to the updates at any end device and server. Since a feasible assignment exists as proven earlier, all m layers are eventually offloaded to servers, thereby leading to a total of $m\lceil\Delta/\epsilon\rceil$ increments. Second, the maximum number of requests between price increments corresponds to the number of iterations for price updates, which are at most $Y \leq mn$. The considerations above apply to both the forward and reverse auction, resulting in a time complexity of $\mathcal{O}(mY\lceil\Delta/\epsilon\rceil)$. □

C. Load Balancing

The multiple assignment problem investigated so far has a weighted sum as the objective function, thereby resulting in a linear programming problem. This section extends the previously-obtained results to the case of load balancing, expressed in terms of the widely-used *proportional fairness*, namely, the sum of the logarithms of utilities (i.e., benefits) [32].

Problem 3 (Multiple Assignment with Proportional Fairness)**.** The multiple assignment problem with proportional fairness has the following form:

$$\max_{\boldsymbol{x},\boldsymbol{y}} \quad \sum_{(i,j)\in\mathcal{Y}} \log(a_{jk}y_{jk})x_{jk} \tag{10a}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{S}(j)} x_{jk} = 1, \quad \forall i \in \mathcal{L}\,, \tag{10b}$$

$$\sum_{j\in\mathcal{L}(k)} y_{jk} \leq 1, \quad \forall j \in \mathcal{S} \tag{10c}$$

$$x_{jk} \in \{0,1\}, \quad \forall (j,k) \in \mathcal{Y} \tag{10d}$$

$$y_{jk} \geq 0, \quad \forall (j,k) \in \mathcal{Y}\,, \tag{10e}$$

Where $0 \triangleq 0 \cdot \log 0$ by definition. The non-negative variable y_{jk} is the resource sharing fraction for layer j at server k, with $\sum_{j\in\mathcal{L}(k)} y_{jk} \leq 1$ for all servers.

Note that the multiple assignment problem with proportional fairness is a mixed optimization problem, as $\boldsymbol{y}$ is a vector

of real variables. The following establishes a correspondence between Problem 3 and Problem 2, so that DAMA can be applied for load balancing too.

The main idea is that the optimal solution to Problem 3 in the vector $\boldsymbol{y}$ is the one that gives a uniform resource allocation [33]. Accordingly, $y^*_{jk} = 1/n_k$ where n_k is the number of users assigned to server j. As a consequence, Problem 3 can be transformed into the following one:

$$\max_{\boldsymbol{x},\boldsymbol{n}} \quad \sum_{(j,k)\in\mathcal{Y}} x_{jk}\log a_{jk} - \sum_{k\in\mathcal{S}} n_k \log n_k \tag{11a}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{L}(k)} x_{jk} = n_k, \quad \forall k \in \mathcal{S}\,, \tag{11b}$$

$$\sum_{k\in\mathcal{S}(j)} x_{jk} = 1, \quad \forall j \in \mathcal{L}\,, \tag{11c}$$

$$x_{jk} \in \{0,1\}, \quad \forall (j,k) \in \mathcal{Y}\,, \tag{11d}$$

which is a non-linear integer optimization problem – thus, also NP hard [31] – where $\boldsymbol{n}$ is a new vector of integer variables.

However, the following observation allows to find an approximate solution to the problem. The optimal assignment in the optimization problem above ensures the proportional fairness among the utilities of the layers. In other words, the corresponding solution also balances the assignments (i.e., loads) among the servers. Accordingly, Problem 2 can be leveraged to approximate the optimization problem with proportional fairness, where the upper and the lower bounds of n_k can be estimated based on the characteristics of DAMA. The rest of the analysis relies on the following notation:

$$u(\boldsymbol{x},\boldsymbol{n}) = \sum_{(j,k)\in\mathcal{Y}} x_{jk}\log a_{jk} - \sum_{k\in\mathcal{S}} n_k \log n_k = u_x(\boldsymbol{x}) + u_n(\boldsymbol{n})$$

Proposition 3. *Let $u(\widetilde{\boldsymbol{x}}^*, \widetilde{\boldsymbol{n}}^*)$ be the value of the objective function for the assignment obtained by DAMA and $u(\boldsymbol{x}^*, \boldsymbol{n}^*)$ the value of the objective function for the optimal solution to the optimization problem in Eqs.* (11a)–(11d)*. If $n_k^- \le n_k^* \le n_k^+$ for all servers k, then the following holds:*

$$u(\boldsymbol{x}^*, \boldsymbol{n}^*) - u(\widetilde{\boldsymbol{x}}^*, \widetilde{\boldsymbol{n}}^*) \le m(\log n + \epsilon)\,.$$

Proof. The proof first shows that the assignment obtained by DAMA is within $m\epsilon$ of the optimal solution then derives the bound for the reformulated version of Problem 3 accordingly.

DAMA terminates due to Proposition 2. It is then enough to show that the ϵ-CS conditions in Definition 1 hold upon termination, as the optimality bound trivially follows from Proposition 1. Consider the ϵ-CS conditions in Eqs. (7) and (8). The following shows that, if they are satisfied at the beginning of an iteration, they are also satisfied at the end of that iteration. Specifically, let (π_j, p_k) be the pair denoting the benefit associated with layer j and the price of server k at the beginning of an iteration for the forward auction. Assume that server k receives a bid for layer j, previously associated with j_k, during the iteration, leading to the updated pair (π'_j, p'_k). Then, $\pi' + p'_{j_k} = a_{jk}$ holds due to line 10 of Algorithm 1. For every other layer $l \in \mathcal{L}$ with $l \neq i$, $\pi' + p'_{j_k} \ge a_{jk} - \epsilon$ holds since $\pi'_l = \pi_l$ and $p'_{j_k} \ge p_{j_k}$ for each server j_k. A similar reasoning applies to the reverse auction. It now remains to verify that the condition in Eq. (9) holds as well. Recall that only servers with less than n_k^- clients participate in the reverse auction. As a consequence, upon termination there are some unassigned virtual servers in $\mathcal{S}^+$ with a zero price, which is at most the price of assigned virtual servers in the same set. Therefore, also the last ϵ-CS condition is satisfied and the assignment obtained by DAMA is within $m\epsilon$ of the optimal solution.

As a result, $u_x(\boldsymbol{x}^*) - u_x(\widetilde{\boldsymbol{x}}^*) \le m\epsilon$, since $n_k^- \le n_k^* \le n_k^+$ by hypothesis. Consequently, it is:

$$\begin{aligned} u(\boldsymbol{x}^*, \boldsymbol{n}^*) - u(\widetilde{\boldsymbol{x}}^*, \widetilde{\boldsymbol{n}}^*) &= u_x(\boldsymbol{x}^*) - u_x(\widetilde{\boldsymbol{x}}^*) + u_n(\boldsymbol{n}^*) - u_x(\widetilde{\boldsymbol{n}}^*) \\ &\le m\epsilon + u_n(\boldsymbol{n}^*) - u_x(\widetilde{\boldsymbol{n}}^*) \\ &= m\epsilon + \sum_{k\in\mathcal{S}} n_k^* \log n_k^* - \sum_{k\in\mathcal{S}} \widetilde{n}_k^* \log \widetilde{n}_k^* \\ &\le m\epsilon + m\log m - m\log\frac{m}{n} \\ &= m(\epsilon + \log n) \end{aligned}$$

□

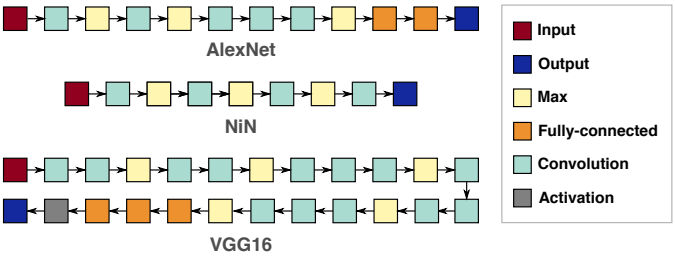


Fig. 2: Architectures of the DNN benchmarks considered in the evaluation: AlexNet, NiN and VGG16.

IV. Evaluation

Experiments are carried out with a custom python network simulator built on top of the PyTorch [34] framework. Three different datasets are considered: Berkeley Deep Drive (BDD100k), containing 120M images from 100K videos captured by cameras on self-driving cars [35]; Stanford Cars (CARS), including 16,185 images for 196 classes of cars [36]; and Canadian Institute for Advanced Research-100 (CIFAR-100), with 60K images divided into 100 classes [37]. DNN tasks are assumed to be independent from each other; they are generated by the devices according to a Poisson distribution with a mean of λ_j. Three widely-used DNN models for image classification are employed as benchmarks: NiN, VGG16, and AlexNet (Fig. 2); the thresholds τ_{ijk} for local execution are derived accordingly.

The deployment area of the network is a square region of 500 m^2. The number of devices is three times the number of edge servers, according to [17, 30, 38]. Edge servers and devices are static and randomly placed in the deployment area according to a uniform distribution; all edge servers have the same computing power. Table II reports the parameters employed in the simulation.

In addition to DAMA, the following offloading schemes for DNN inference are considered for comparison purposes.

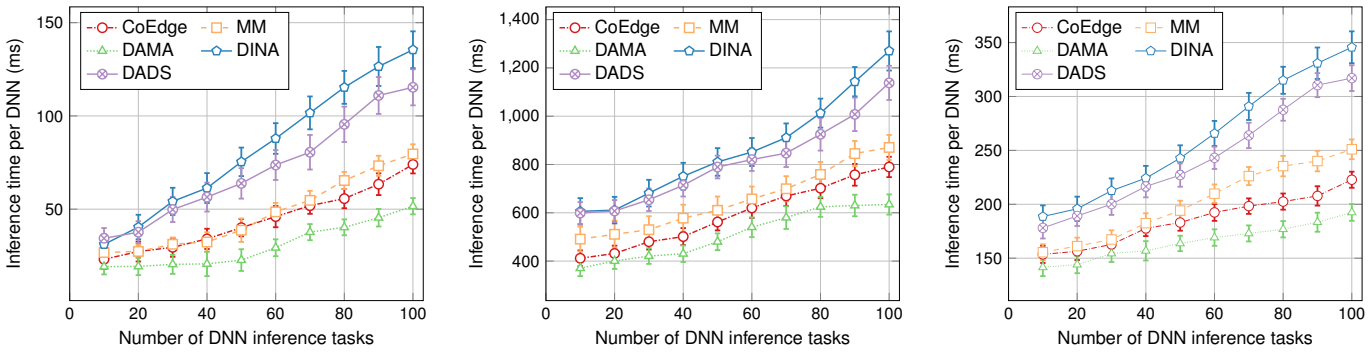


Fig. 3: Total inference time of the different schemes as a function of the DNN tasks for (a) NiN, (b) VGG16, and (c) AlexNet.

TABLE II
SIMULATION PARAMETERS

Parameter	Value
Deployment area	500×500 m
Transmission range	30 m [40]
Bandwidth (B)	22.75 Mbps [16]
Number of edge servers (n)	30 [17, 30, 38]
Number of end devices (D)	90 [17, 30, 38]
Mean task arrival rate (λ_j)	8 tasks/s [17]
Processing density (c)	8 double-precision FLOPS/cycle [41]
Edge compute power (ν)	10^{10} cycle/s [30]
Task delay threshold (τ_{ijk})	[5, 110] ms [17]

- Minmax (MM), a modified version of Problem 1 that minimizes the maximum execution time of all the tasks in the network.
- DADS, the partitioning scheme under light workload in [16], based on the minimum weighted s-t cut of a DNN.
- DINA, the layer-wise decomposition in [17], based on swap matching under two-sided stability.
- CoEdge, the horizontal partitioning scheme in [18], which minimizes the energy cost of offloading subject to latency constraints.

It is worth noting that all the schemes above excluding DADS are distributed.

Offloading all DNN tasks is determined upon their arrival at end devices for all schemes and intermediate data are transmitted directly between edge servers. Furthermore, the following parameters are employed for both DAMA and MM, similar to [39]: $\epsilon = 0.01$, $n_k^+ = \lceil 3m/n \rceil$, and $n_k^- = \lceil 0.75m/n \rceil$.

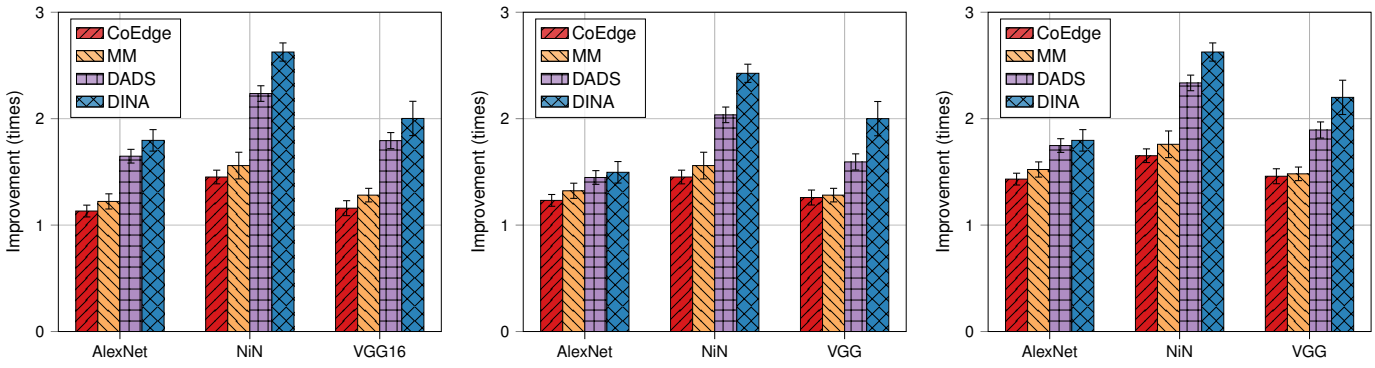
A. Simulation Results

The evaluation considers the following performance metrics: inference time, the improvement over the state of the art, load balancing as well as fairness, and convergence[7]. Unless otherwise stated, results refer to the BDD100k dataset and individual data points are the average of twenty replications for each experiment, with error bars representing the corresponding standard deviations.

Inference Time. Fig. 3 shows the average inference time (i.e., the total execution time T_{ijk}) as a function of the number of tasks for all the considered schemes (i.e., CoEdge, DAMA, DINA, DADS, MM) and architectures (NiN, AlexNet, and VGG16). DAMA obtains the lowest inference time in all cases, followed by CoEdge and MM. The total inference time of DAMA, CoEdge, and MM clearly grows with the number of inference tasks, even though it never doubles across consecutive values of the considered parameters. In contrast, the inference time of both DADS and DINA increases more significantly. This clearly demonstrates the scalability of the proposed schemes for all the three DNN benchmarks. Clearly, NiN obtains the lowest inference time, always below 150 ms (Fig. 3a), as it has the least number of layers and substantially smaller intermediate outputs than the other benchmarks. Instead, the total inference time for VGG16 is significantly higher than both NiN and AlexNet, exceeding even one second (Fig. 3b), due to the higher number of layers and larger convolutions.

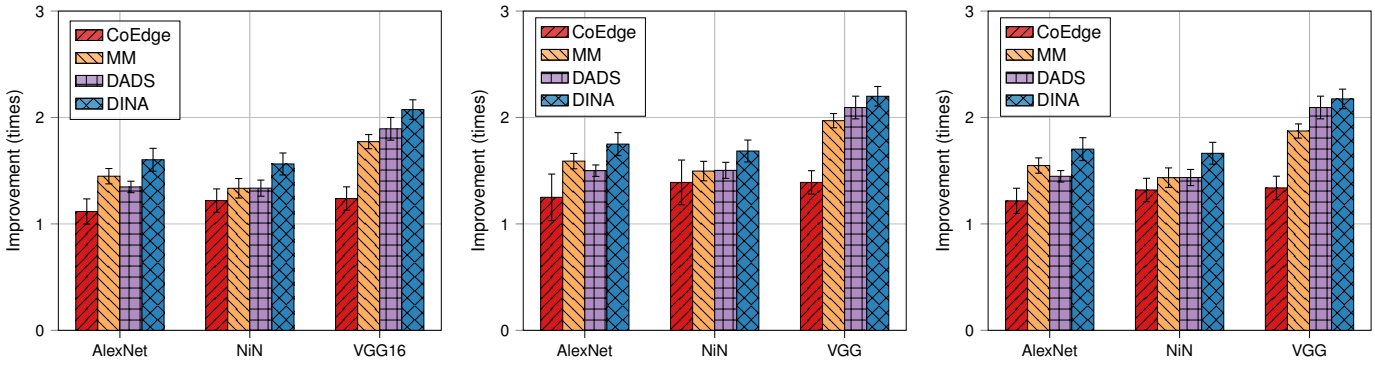
Improvement. Figs. 4-6 show the improvement of the proposed solution as the ratio between the time obtained with a certain scheme and that of DAMA, as a function of the considered DNN benchmarks and for the considered datasets. In particular, Fig. 4 focuses on the total inference time for BDD100K (Fig. 4a), CARS (Fig. 4b), and CIFAR (Fig. 4c). It is clear how DAMA performs much better than the other solutions, with improvements between 1.14 and 2.62 for BDD100K, 1.16 to 2.42 for CARS, 1.45 to 2.68 for CIFAR. This proves that DAMA is effective independent from the considered dataset, while the actual performance varies more significantly based on the specific DNN benchmark, as already shown in Fig. 3. Fig. 5 illustrates the transmission time for the three considered datasets. Also in this case DAMA significantly improves on the other schemes in all cases, even though the difference between the minimum and maximum gains are smaller than for the total inference time. This happens as intermediate data are directly transferred between edge servers for offloaded tasks in all the schemes. CoEdge performs better than the other approaches as it partitions layers horizontally, therefore, it only requires exchanging data at the very beginning and at the end of the offloading process to combine the output. Moreover, DADS performs similar to MM as it splits the DNN into two partitions only, thereby limiting the amount of data to be transferred. The improvements for CARS (Fig. 5b) and CIFAR100 (Fig. 5c) are slightly better than for BDD100K

[7]Convergence as considered here refers to the time needed to find an offloading assignment. The related process does not affect model convergence because it employs a pre-trained DNN as it is, without the need for re-training.



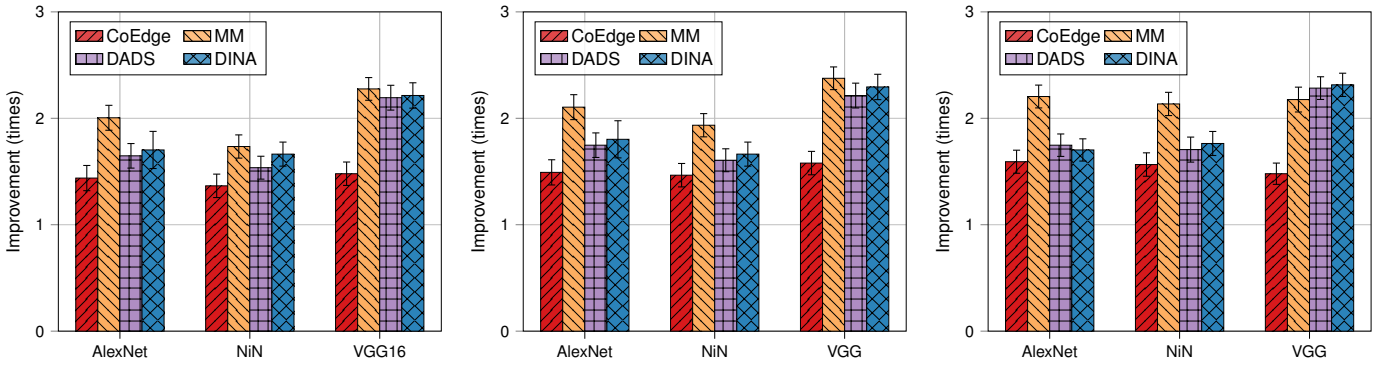
(a) (b) (c)

Fig. 4: Improvement of DAMA against the other schemes in terms of the total inference time for the considered DNN benchmarks with the (a) BDD100K, (b) CARS, and (c) CIFAR100 datasets.



(a) (b) (c)

Fig. 5: Improvement of DAMA against the other schemes in terms of transmission time for the considered DNN benchmarks with the (a) BDD100K, (b) CARS, and (c) CIFAR100 datasets.

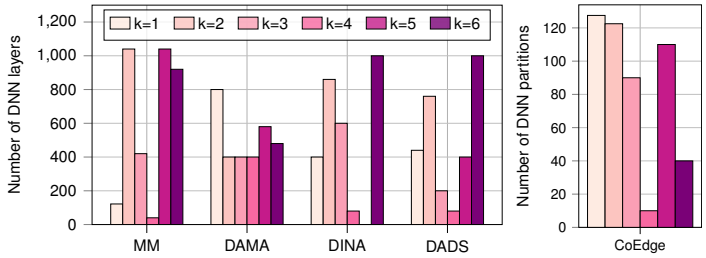


(a) (b) (c)

Fig. 6: Improvement of DAMA against the other schemes in terms of queuing time for the considered DNN benchmarks with the (a) BDD100K, (b) CARS, and (c) datasets.

(Fig. 5a) as the input/output data are smaller. Finally, Fig. 6 characterizes the queuing time for the different datasets. Again, DAMA outperforms the other schemes irrespective of the considered dataset. In particular, the gain over MM is higher than that over both DADS and DINA. This can be explained on the basis of the objective function of MM, which reduces the variance in the computation time by increasing the chance that a task is queued. The difference in the actual results between BDD100K (Fig. 6a), CARS (Fig. 6b) and CIFAR-100 (Fig. 6c) is very small for all the considered schemes.

Load balancing and fairness. Fig. 7 illustrates the load distribution on the individual edge nodes in a network with six servers when using the (most demanding) VGG16 benchmark for all the considered schemes. Due to the different nature of such schemes, Fig. 7a reports load as number of layers for MM, DAMA, DINA and DADS; whereas Fig. 7a shows it as number of partitions for CoEdge, as these are formed across layers. The figure shows an uneven distribution of the layers for MM, DINA, and DADS. In particular, some edge servers receive significantly more DNN layers (even more than 1,000) than others (less than 100). Such a disparity



(a) (b)

Fig. 7: Load distribution for the considered schemes in a network with six edge servers as the number of (a) DNN layers and (b) partitions offloaded to each server.

Scheme	AlexNet	NiN	VGG16
CoEdge	0.8149	0.8478	0.8523
DAMA	0.9061	0.8934	0.9120
DADS	0.7412	0.7281	0.7312
DINA	0.5291	0.5188	0.5312
MM	0.7363	0.7121	0.7148

TABLE III
ASSOCIATION FAIRNESS OF THE CONSIDERED SCHEMES AS A FUNCTION OF THE DIFFERENT DNN BENCHMARKS.

TABLE IV
CONVERGENCE TIME OF THE DIFFERENT SCHEMES FOR THE CONSIDERED DNN BENCHMARKS.

Scheme	AlexNet	NiN	VGG16
CoEdge	10.56	11.22	13.49
DAMA	9.65	11.78	14.25
MM	10.94	12.18	16.05
DINA	78.49	67.56	88.43
Consensus-based	85.23	101.59	120.39

results in a high inference time, as network resources are not fully utilized. In contrast, DAMA achieves the most balanced utilization, with layers almost uniformly distributed across all edge servers. This indeed maximizes the network-wide proportional fairness, implying that all DNN layers are fairly offloaded almost irrespective of how many devices are close to the edge servers. The distribution of partitions in CoEdge is also uneven, as two edge servers (i.e., $k = 4$ and $k = 6$) have much lower load than the others. However, such a distribution is better than that obtained by MM, DINA and DADS – even though it does not reach the level of load balancing in DAMA.

Table III reports the average of Jain's fairness index [32] (between zero and one, the higher the better) calculated for edge server k as $\Xi_k = \frac{(\sum_{i=1}^{|\mathcal{D}|} l_{ik})^2}{|\mathcal{D}|(\sum_{i=1}^{|\mathcal{D}|}(l_{ik})^2)}$, where l_{ik} is the total number of layers from user i associated with server k. Such an index is derived for the three DNN benchmarks in all considered scenarios with respect to the related association of devices. In line with the previous findings, DAMA obtains the highest fairness index of about 0.9, therefore, the most fair allocation. Also here CoEdge performs better than DADS, MM, and DINA due to the horizontal partitioning across the layers. DADS improves over MM and DINA as its partitioning technique results in a lower number of layers to be offloaded per DNN. Notably, the fairness of all schemes does not significantly vary over the different DNN benchmarks.

Convergence. Table IV shows the average convergence time as the average number of iterations needed by the different schemes to terminate. In particular, CoEdge, DAMA, DINA and MM are compared to a consensus-based algorithm [42] for the considered DNN benchmarks. CoEdge is a recursive technique; accordingly, the number of iterations here indicates the recursion depth reached. DAMA and MM require a similar number of iterations (at most sixteen) as MM employs a modified version of the iterative algorithm in DAMA, which has a polynomial time complexity (recall Proposition 2). DAMA obtains the fastest convergence time for AlexNet, while CoEdge the best results for NiN and VGG16; however, the difference is not significant. The number of iterations of both DINA and the consensus-based scheme are almost one order of magnitude higher than the other approaches. This happens as DINA relies on swap matching; as a consequence, each iteration involves a pair of nodes, thereby resulting in a longer time to converge. Similarly, reaching distributed consensus also entails a significant overhead.

B. Summary and Discussion

The obtained results have demonstrated that DAMA significantly improves over the state of the art in terms of inference latency, irrespective of the considered DNN benchmark. It also effectively balances the load across edge servers and converges fast to a near-optimal assignment. These results derive from the expressive and detailed model of the offloading process, combined with its formulation as an assignment problem.

The theoretical premises of this work laid out a solid foundation for load balancing in offloading DDN inference at the edge. However, they also have some limitations. Among them, the communication model considered a shared wireless channel with a certain bandwidth equally divided among end devices /edge servers, similar to [16]. Such an assumption could be easily relaxed by considering a different bandwidth between each pair of nodes, as in [18]. The communication model could also be replaced with one based on rate, as that used in [17].

Handling time-varying changes in the system parameters is more challenging. In fact, an accurate characterization of both communication and computation is necessary to efficiently offload inference due to precedence constraints. Parameters could be continuously estimated and dynamically updated over time at the cost of additional overhead for possibly updating offloading decisions. Varying channel conditions could be more easily accounted for by taking a conservative approach that considers a guaranteed (i.e., minimum) bandwidth. More complex channel models that explicitly incorporate uncertainty, as the one in [43], might be employed as well.

V. RELATED WORK

Several works in the literature have considered DNN inference acceleration through the edge and (or) the cloud. DDNN [44] employs an early-exit strategy – to stop processing

early, as long as a certain accuracy is received – to reduce inference time in a tiered network consisting of end devices, the edge, and the cloud. ADDA [45] and Edgent [46] jointly apply an early-exit strategy and DNN segmentation to perform coordinated device-edge inference. SPINN [47] also combines an early-exit strategy with DNN partitioning but especially targets dynamic conditions under user-defined service agreements. All the works above tradeoff accuracy for inference time and may require special training. In contrast, the approach considered in this article targets pre-trained DNNs and allows distributed inference with no loss in accuracy.

Different works have addressed exact inference with pre-trained DNNs too. Among them, Neurosurgeon [19] partitions and offloads DNN computation between mobile devices and the cloud. Specifically, it offloads layers in a linear DNN to the cloud by minimizing both latency and energy consumption, based on real-time monitoring of network traffic. IONN [24] divides a DNN layer-wise into multiple partitions and incrementally offload them to edge nodes to reduce both transmission and computation time through parallel execution. In doing so, it applies the shortest path algorithm on the graph-based characterization of the DNN. Shin et al. [48] extend the IONN partition technique to obtain a fine-grained partitioning scheme based on an efficiency metric, defined as the ratio between the latency reduction and the transmission overhead. DADS [16] leverages a graph-theoretic approach and splits a DNN into two partitions, one offloaded to the cloud and the other to the edge. In particular, DADS determines two types of graph cuts: one to minimize latency for light workloads and another to maximize throughput for heavy workloads. The solutions mentioned above leverage centralized algorithms; instead, the solution developed here is distributed.

Finally, a few works have explicitly targeted exact distributed DNN inference. MoDNN [49] considers mobile devices connected through high-rate WiFi links forming a local cluster. Specifically, MoDNN aims at reducing the synchronization overhead of the devices as they collectively carry out inference tasks through different DNN partitioning schemes. DINA [17] introduces an adaptive partitioning algorithm to split a DNN with sub-layer granularity. It also includes a distributed offloading technique based on matching theory, namely, a swap-matching algorithm that targets reducing the total inference time. DeepThings [50] devises a partitioning technique that fuses the feature maps in convolutional layers to enable their synchronized execution at multiple edge nodes through a work-stealing algorithm. Similarly, CoEdge [18] applies an adaptive technique that partitions convolutional layers horizontally and offloads them to edge nodes. Moreover, it carries out cooperative inference by minimizing the energy cost for both computation and communication, subject to deadline constraints. CoopAI [51] employs dynamic programming for partitioning multiple layers, which are grouped into a block and processed in a round. Edge devices work together on the blocks and the intermediate results are obtained by prefetching extra data. EDGE-LD [52] leverages a MapReduce paradigm to divide a DNN workload among heterogeneous edge devices based on a profiling model that considers both execution time and used bandwidth. A layer fusion scheme is also proposed to reduce the communication overhead. Despite being distributed, all these solutions either leverage heuristics or frameworks that do not provide strong optimality guarantees. In contrast, the theoretical framework developed in this work achieves near-optimal DNN inference offloading at the edge.

Distributed assignment problems have been extensively investigated in other scenarios, including wireless networking and distributed computing [39, 53, 54]. Xu et al. [39] propose a distributed auction algorithm to decide how to associate clients to access points in mm-wave communication. Lee et al. [53] develop a framework based on deep learning to solve constrained optimization problems in a distributed fashion, then apply it to resource allocation in wireless networks. Castellano et al. [54] introduce a distributed algorithm based on max-consensus to assign resources to applications in an edge infrastructure. In contrast, this article is the first to model and optimally solve the problem of offloading inference with pre-trained DNNs in the context of edge computing.

VI. Conclusion

This article presented a model and an optimization problem to offload DNN inference in edge networks with load balancing. In particular, a detailed model of the offloading process was devised and the corresponding problem was formulated as a multiple assignment that maximizes proportional fairness. A distributed auction algorithm (DAMA) was introduced accordingly; its optimality as well as time complexity were also analytically characterized. Results from extensive simulations in realistic settings demonstrated that DAMA is more efficient than the state of the art, achieving near-optimal performance according to the analytically-derived bounds. In particular, DAMA obtains the lowest total inference time for widely-used DNN benchmarks, as well as the highest fairness with a Jain's index of 0.9. Moreover, DAMA requires only a very limited number of iterations to offload DNN layers to edge servers. A promising future work is represented by evaluating DAMA in an edge testbed with heterogeneous resource-constrained devices and by considering sub-layer partitioning of DNNs. Ensuring security and privacy of offloading through distributed ledger technologies is also another interesting research direction.

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] H. Yang, A. Alphones, Z. Xiong, D. Niyato, J. Zhao, and K. Wu, "Artificial-intelligence-enabled intelligent 6G networks," *IEEE Network*, vol. 34, no. 6, pp. 272–280, 2020.

[4] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[5] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient

inference and training in neural networks," *Journal of Machine Learning Research*, vol. 22, no. 241, pp. 1–124, 2021.

[6] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[7] Z. Lian, W. Wang, and C. Su, "COFEL: Communication-efficient and optimized federated learning with local differential privacy," in *ICC 2021 - IEEE International Conference on Communications*. IEEE, 2021.

[8] Y. Liu, Y. Zhu, and J. J. Yu, "Resource-constrained federated learning with heterogeneous data: Formulation and analysis," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2021.

[9] H. Yang, J. Zhao, Z. Xiong, K. Lam, S. Sun, and L. Xiao, "Privacy-preserving federated learning for UAV-enabled networks: Learning-based joint scheduling and resource management," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 10, pp. 3144–3159, 2021.

[10] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4921–4934, 2019.

[11] M. Asad, A. Moustafa, T. Ito, and M. Aslam, "Evaluating the communication efficiency in federated learning algorithms," 2020.

[12] F. Sattler, S. Wiedemann, K.-R. Muller, and W. Samek, "Robust and communication-efficient federated learning from non-i.i.d. data," *IEEE Transactions on Neural Networks Learning Sys.*, vol. 31, no. 9, pp. 3400–3413, 2020.

[13] W. Wang, M. H. Fida, Z. Lian, Z. Yin, Q.-V. Pham, T. R. Gadekallu, K. Dev, and C. Su, "Secure-enhanced federated learning for AI-empowered electric vehicle energy prediction," *IEEE Consumer Electronics Magazine*, 2021.

[14] 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, 2017.

[15] Q.-H. Nguyen and F. Dressler, "A smartphone perspective on computation offloading – a survey," *Computer Communications*, vol. 159, pp. 133–154, 2020.

[16] 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.

[17] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *The 39th IEEE International Conference on Computer Communications (INFOCOM 2020)*, Jul. 2020, pp. 854—863.

[18] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2021.

[19] 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.

[20] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[21] Z. Xiong, S. Feng, D. Niyato, P. Wang, and Z. Han, "Optimal pricing-based edge computing resource management in mobile blockchain," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018.

[22] Y. Zhang, T. Scargill, A. Vaishnav, G. Premsankar, M. Di Francesco, and M. Gorlatova, "InDepth: Real-time depth inpainting for mobile augmented reality," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 1, pp. 1–25, March 2022.

[23] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the internet of things: A case study," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275–1284, April 2018.

[24] H. Jeong, H. Lee, C. H. Shin, and S. Moon, "IONN: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2018.

[25] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Belmont, Massachusetts: Athena Scientific, 1998.

[26] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[27] L. Massoulié and J. Roberts, "Bandwidth sharing: objectives and algorithms," in *Proc. IEEE International Conference on Computer Communications (INFOCOM '99)*, vol. 3, 1999, pp. 1395–1403.

[28] "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.

[29] 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.

[30] 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.

[31] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.

[32] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *Digital Equipment Corp., Tech. Rep.*, 1998.

[33] Q. Ye, B. Rong, Y. Chen, M. Al-Shalash, C. Caramanis, and J. G. Andrews, "User association for load balancing in heterogeneous cellular networks," *IEEE Trans. Wireless Commun.*, vol. 12, no. 6, pp. 2706–2716, Jun. 2013.

[34] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[35] 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.

[36] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[37] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[38] 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.

[39] Y. Xu, H. S. Ghadikolaei, and C. Fischione, "Adaptive distributed association in time-variant millimeter wave networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 1, pp. 459–472, Jan 2019.

[40] B. Jedari and M. Di Francesco, "Delay analysis of layered video caching in crowdsourced heterogeneous wireless networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.

[41] R. Dolbeau, "Theoretical peak flops per instruction set: a tutorial," *The Journal of Supercomputing*, vol. 74, no. 3, pp.

1341–1377, Mar 2018.

[42] L. B. Johnson, H. L. Choi, and J. P. How, "The role of information assumptions in decentralized task allocation: A tutorial," *IEEE Control Systems*, vol. 36, no. 4, pp. 45–58, Aug 2016.

[43] H. Yang, Z. Xiong, J. Zhao, D. Niyato, L. Xiao, and Q. Wu, "Deep reinforcement learning-based intelligent reflecting surface for secure wireless communications," *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 375–388, 2021.

[44] 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.

[45] H. Wang, G. Cai, Z. Huang, and F. Dong, "ADDA: Adaptive distributed DNN inference acceleration in edge computing environment," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2019.

[46] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Commun.*, vol. 19, no. 1, pp. 447–457, 2020.

[47] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. ACM, 2020.

[48] K. Y. Shin, H. Jeong, and S. Moon, "Enhanced partitioning of DNN layers for uploading from mobile devices to edge servers," in *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications - EMDL '19*. ACM Press, 2019.

[49] 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.

[50] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Transactions on Computer-Aided Design Integrated Circuits Sys.*, vol. 37, no. 11, pp. 2348–2359, 2018.

[51] C. Yang, J. Kuo, J. Sheu, and K. Zheng, "Cooperative distributed deep neural network deployment with edge computing," in *ICC 2021 - IEEE International Conference on Communications*. IEEE, 2021.

[52] F. Xue, W. Fang, W. Xu, Q. Wang, X. Ma, and Y. Ding, "EdgeLD: Locally distributed deep learning inference on edge device clusters," in *2020 IEEE 22nd International Conference on High Performance Computing and Communications;IEEE 18th International Conference on Smart City;IEEE 6th International Conference on Data Science and Systems*. IEEE, 2020.

[53] H. Lee, S. H. Lee, and T. Q. Quek, "Deep learning for distributed optimization: Applications to wireless resource management," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2251–2266, 2019.

[54] G. Castellano, F. Esposito, and F. Risso, "A distributed orchestration algorithm for edge computing resources with guarantees," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2548–2556.