Efficient Distributed DNNs in the Mobile-edge-cloud Continuum

Efficient Distributed DNNs in the Mobile-edge-cloud Continuum

Francesco Malandrino, *Senior Member, IEEE,* Carla Fabiana Chiasserini, *Fellow, IEEE,* Giuseppe Di Giacomo

***Abstract*—In the mobile-edge-cloud continuum, a plethora of heterogeneous data sources and computation-capable nodes are available. Such nodes can cooperate to perform a *distributed* learning task, aided by a learning controller (often located at the network edge). The controller is required to make decisions concerning (i) data selection, i.e., which data sources to use; (ii) model selection, i.e., which machine learning model to adopt, and (iii) matching between the layers of the model and the available physical nodes. All these decisions influence each other, to a significant extent and often in counter-intuitive ways. In this paper, we formulate a problem addressing all of the above aspects and present a solution concept called RightTrain, aiming at making the aforementioned decisions in a joint manner, minimizing energy consumption subject to learning quality and latency constraints. RightTrain leverages an expanded-graph representation of the system and a delay-aware Steiner tree to obtain a provably near-optimal solution while keeping the time complexity low. Specifically, it runs in polynomial time and its decisions exhibit a competitive ratio of $2(1+\epsilon)$, outperforming state-of-the-art solutions by over 50%. Our approach is also validated through a real-world implementation.**

***Index Terms*—Distributed machine learning; mobile-edge-cloud continuum; 5G and beyond networks.**

I. Introduction

Enabling technologies like 5G networks and distributed machine learning (ML) have fostered the emergence of the so-called *Internet of Intelligent Things* networking paradigm, allowing user equipment (UEs), e.g., smartphones or smart-city actuators, to leverage cloud-based artificial intelligence services. This scenario is expected to further evolve towards the edge intelligence paradigm [2], [3]: ML-based applications will move from remote, cloud-based servers to the mobile network, including computation-capable devices at the network edge and mobile UEs. Indeed, recent reports [4] highlight how the capability of edge and mobile devices is growing much faster than cloud ones, soon leading to complete interoperability among mobile, edge, and cloud and, thus, to the formation of a *continuum*.

The ML tasks to perform will be as diverse as the devices performing them. Indeed, the best ML approach to adopt depends upon such factors as the application and scenario at hand, as well as the type and quantity of available data. Possible options include supervised [5]–[7] and unsupervised learning (most notably, deep domain adaptation (DDA) [8]–[10]), as well as hybrid applications combining labeled and unlabeled data through multiple learning and pseudo-labeling techniques [11]. Many of these tasks can be accomplished through deep neural networks (DNNs), built by combining a sequence of *layers* of different types. The DNNs used for unsupervised and hybrid learning tend to be more complex than their supervised learning counterparts, however, they use the very same building blocks, e.g., convolutional and fully-connected layers [5], [7]. The possibility of using a relatively small set of building blocks to support a wide set of ML-based applications has motivated the ML-as-a-Service (MLaaS), whereby the network provides a set of *customized* ML-based services, e.g., image recognition or clustering, to the applications using the network resources.

Owing to the complexity of the learning tasks to perform, as well as to the need to keep as much as possible the information coming from different sources local, it is expected that most of MLaaS learning will be performed in a *distributed* fashion. Distributed learning is an excellent fit for edge intelligence scenarios, as it envisions leveraging the data and resources of multiple nodes in order to perform a common learning task. Additionally, recent works on ML techniques tailored for lower-powered devices [12], [13] has significantly extended the types of devices that can be leveraged for distributed ML, along with their diversity. We thus argue that ML-based services can seamlessly use any of the nodes of the mobile-edge-cloud continuum, so as to ensure that requirements (e.g., learning time) are met.

Among the factors influencing the performance of MLaaS, combined with distributed learning, the most prominent are (i) the quantity and diversity of the input data used for training; (ii) the actual learning strategy employed, e.g., the layers composing the DNN; (iii) the resources, (e.g., computational) available to the learning task. Such decisions strongly impact one another, e.g., using more data requires more resources to promptly process them; thus, it is important that they are made in a joint manner. The split learning (SL) paradigm [14]–[16], envisioning to run a subset of the DNN layers at each level of the network topology, represents a significant step in this direction. However, SL is concerned only with placement decisions and, since it only splits DNN into as many parts as there are network topology layers (e.g., one for edge and one for cloud), may not always be able to reap the full benefits of the mobile-edge-cloud continuum.

Another limit of state-of-the-art works on distributed ML is their emphasis on learning *effectiveness*, e.g., classification accuracy, over *efficiency*. Indeed, most existing approaches are concerned with locating and selecting the highest possible quantity of computational resources, and use them to process the largest possible quantity of data, in order to obtain the

F. Malandrino and C. F. Chiasserini are with CNR-IEIIT and CNIT, Italy. C. F. Chiasserini and G. Di Giacomo are with Politecnico di Torino, Italy.

A preliminary version of this work was presented in [1].

highest-quality possible learning. However, awareness is rising that seeking the utmost performance is not necessarily the most desirable strategy in real-world situations, and the concept of ML efficiency – as opposed to sheer performance – is rapidly gaining prominence [17]. In fact, while *inference* has a relatively minor impact on the total resource consumption, the *training* of a DNN is very demanding in terms of processing power, hence, energy.

In this paper, we address the above issues by presenting RightTrain, a decision-making framework allowing joint, high-quality decisions on (i) which data to use for learning, (ii) which DNN structure to employ, and (iii) which physical nodes and resources therein to use. Our framework can capture the nontrivial (and, often, counter-intuitive) ways in which such choices interact with each other, and yields decisions that are provably close to the optimum, while keeping a low computational complexity.

More specifically, our contributions can be summarized as follows:

- we propose a model that describes the components of a MLaaS system based on DNNs, their behavior, and their inter-dependencies;
- we formulate the problem of *jointly* making decisions on: (i) the data to be used for model training, (ii) the structure of the DNN to adopt, and the resources to allocate for the learning process, with the aim of minimizing the energy consumption while meeting a target maximum learning time and desired learning quality;
- we present a solution, called *RightTrain*, solving the above problem in polynomial time and yielding provably near-optimal (namely, with a $2(1+\epsilon)$ competitive ratio) solutions;
- we compare RightTrain against the SL state-of-the-art approach, and find how the greater flexibility of RightTrain results in a significantly lower energy consumption, especially when the target learning time is not too short;
- we show the feasibility of our approach using a lab test-bed implementation.

As discussed in Sec. II, several existing works have tackled the problem of selecting the computational and network resources needed for a given learning task, and a few have studied the impact of different DNN structures on the learning performance. Our work is, however, the first to identify and solve the important challenge of *adapting* the DNN structure, the network resources and the size of the datasets to one another, thus achieving unparalleled learning efficiency and performance.

The remainder of this paper is organized as follows. After discussing related work in Sec. II and summarizing our main results in Sec. III, we describe our system model and problem formulation in Sec. IV and our characterization of learning performance in Sec. V. We then introduce the RightTrain solution concept in Sec. VI, and formally prove its complexity and competitive ratio properties in Sec. VII. Finally, Sec. VIII shows the performance of RightTrain against the state-of-the-art, Sec. IX reports how we validate our model and approach through a lab test-bed, and Sec. X concludes the paper.

II. Related Work

A first body of works related to ours [18]–[20] target the problem of characterizing the performance of distributed ML, accounting for such aspects as the topology of the *logical* network formed by cooperating nodes [18] and the computational [19], [20] and communication [20] resources they are assigned.

Concerning distributed learning techniques themselves, one of the most prominent is federated learning (FL) [21], whereby participating nodes train the same DNN with their local data, and send the resulting weights to a coordination server that averages the weights and sends them back to the nodes. FL has become one of the most popular approaches to cooperative learning also in mobile [22] and MEC-based [19] scenarios. Since nodes are expected to contribute their own resources to learning, incentive mechanisms may be necessary to foster cooperation [23], also employing blockchain [24].

Several recent works have endeavored to characterize and improve the performance of FL under realistic conditions, most notably, learning nodes with heterogeneous datasets and/or capabilities. The authors of [25] consider the classic FedAvg strategy, and characterize its performance, e.g., the loss reduction as a function of the number of local epochs and global iterations. The later work [26] aims at going beyond FedAvg and proposes an alternative strategy called FedUN. FedUN optimizes the convergence speed by choosing ("sampling") the learning nodes to use at each epoch – and weighting their updates – based upon local gradient information. [26] also provides a lower bound for FedUN's loss reduction at each epoch, hence, the total convergence time. [27] takes a different approach and tackles the issue of fairness, i.e., how to ensure that devices with different capabilities and/or datasets have similar learning performance. The strategy envisioned in [27] is predicated upon giving *more* weight to updates from the nodes with the worst performance, resulting in a better learning quality for such nodes at the cost of a higher number of global iterations.

Split learning (SL) is a recently-emerging paradigm predicated on partitioning the DNN among the nodes participating in the learning process. SL drops FL's requirement that all nodes have the same DNN, and has been found to outperform ML in a wide variety of scenarios [16], owing to its ability to match the learning operations and the hardware performing them. Other works envision similar approaches, based on choosing the right network node to run each layer of a DNN and accounting for device capability [28], network latency [28], [29], and privacy [28], [30]. [29] takes a further step, combining DNN splitting with "right-sizing", i.e., removing some DNN layers if they are not necessary to reach the required learning quality. However, that work only focuses on inference, and neither the quantity of data to use nor the resources to assign are accounted for.

Among the few works jointly making learning- and network-related decisions, [31] aims at (i) right-sizing the DNN for the task at hand, i.e., skipping some layers if the classification precision is sufficiently high, and (ii) offloading a part of the DNN to edge-based nodes. Notice, however,

that [31] only supports DNNs with a chain topology, and does not support the use of multiple sources of information. Still in the context of inference, the authors of [32] envision partitioning the DNN into an arbitrary number of parts, and running each of them at the most appropriate node in the edge-cloud continuum. In a similar setting, [33] addresses security issues, placing different layers of an image-classification DNN on different devices, preventing any device from seeing enough layers to reconstruct the original image.

Recent work [34] targets a heterogeneous scenario similar to the mobile-edge-cloud continuum, and envisions an *inference delivery network* whereby each node offers one or more ML models. Inference tasks – which are assumed to require one of several alternative models, e.g., DNNs with different architectures – can then be carried out at the most appropriate node, accounting for both cost and delay issues. Compared to [34], our work (i) targets the *learning* phase, which is the most challenging and resource-intensive; (ii) allows breaking down *one* model (e.g., one DNN) across multiple devices, and (iii) does not depend upon the assumption that alternative models exist for a given task.

Our work is also related to the recent but growing body of works accounting for ML energy consumption, and the resulting carbon footprint. As reported in [35], training one complex ML model may lead to a carbon footprint equivalent to 5 times the lifetime emissions of an average car. Thus, it is critical to envision solutions that, exploiting the edge intelligence concept [3], [36], effectively exploit the physical proximity of a large number of devices, each collecting data and equipped with computational and memory resources. At a higher level, as advocated in [17], this calls for a different view of ML goals where the focus shifts from sheer learning quality (e.g., classification accuracy) to the more comprehensive concept of *efficiency*.

Finally, a preliminary version of RightTrain has been presented in our conference paper [1]. Compared to [1], this paper includes a more rigorous discussion of RightTrain and the principles it is based upon, a formal characterization of its computational complexity and competitive ratio, additional performance evaluation scenarios, and a testbed-based validation.

Novelty. Our holistic approach contributes to making ubiquitous ML reality, *jointly* addressing issues that earlier have been only marginally or incidentally dealt with. Specifically, we account for the mutual influence of the decisions on (i) the data used for DNNs training, (ii) the DNN structure employed, and (iii) the physical nodes running the latter. Accounting for all aspects and making all decisions jointly allows us to reach a level of effectiveness *and* efficiency that cannot be matched by existing approaches.

III. Main Results and Roadmap

Our first major contribution is represented by the system model and problem formulation summarized in Fig. 1 and described in Sec. IV. The system model accounts for all the main entities and aspects involved in distributed training of DNNs over the mobile-edge-cloud continuum. It can describe ML approaches leveraging data parallelism, model parallelism, or a combination of both – including split learning [14], [15]. The problem formulation then formalizes how the decisions of (i) selecting the input data to be leveraged for learning, (ii) choosing the DNN structure to be used, and (iii) matching DNN layers with physical servers influence each other, and determine the learning efficiency under the constraints imposed by both the learning process and the network system. The problem is shown to be NP-hard.

Next, through a combination of existing measurements taken from the literature and our own experiments, we assess how such decisions impact the learning performance, namely, learning time, learning quality, and energy consumption (Sec. V). Importantly, so doing, we bridge the gap between the abstract system model of Sec. IV and actual, real-world distributed ML solutions.

Building upon the above results and in light of the problem complexity, we envision a solution concept, also summarized in Fig. 3. Its main component is the RightTrain algorithm, detailed in Alg. 1 (Sec. VI), which leverages expanded graphs such as the one shown in Fig. 4 and applies a delay-aware Steiner tree on such graph, to make near-optimal decisions on data selection, DNN structure, and layer-to-node matching. More specifically, as proven in Sec. VII, our solution strategy has polynomial worst-case time complexity (Property 3) and a competitive ratio of $2(1+\epsilon)$. Finally, our performance evaluation shows that the proposed solution reduces by 50% the energy consumption of a learning task when compared to the state of the art, while our lab test-bed implementation shows its feasibility.

IV. System Model and Problem Formulation

Our system model describes a DNN training task leveraging distributed learning, and exploiting the resources of multiple mobile, edge, and cloud nodes (hereinafter also referred to as learning nodes). Such nodes are coordinated by a *central controller*, typically running at the edge of the network infrastructure, which can communicate with all learning nodes and collects information on their capabilities and position. The entities the model represents, along with the decisions the central controller has to make, are depicted in Fig. 1.

A. Input information

Under the DNN paradigm, learning tasks are performed by a set of *layers* of different types (e.g., fully-connected or convolutional), organized as a *tree*: the learning result is the output of the tree root, while leaves correspond to data sources. Each layer has a local set of *parameters* that define its behavior, e.g., the weights of a fully-connected layer, and *training* a DNN means finding the parameter values that minimize a global error function. As an example, for a classification task, the learning output $\mathbf{y}$ represents the probabilities associated with each class, and a typically-used loss function is cross-entropy, defined as $f(\hat{\mathbf{y}}, \mathbf{y}) = \hat{\mathbf{y}}^{\mathsf{H}} \log \hat{\mathbf{y}}$, where $\hat{\mathbf{y}}$ is a column vector containing the ground truth, and $^{\mathsf{H}}$ represents the transpose operator. Training proceeds in an iterative fashion [37] through several *epochs*, each including (i) a *forward pass*, where the

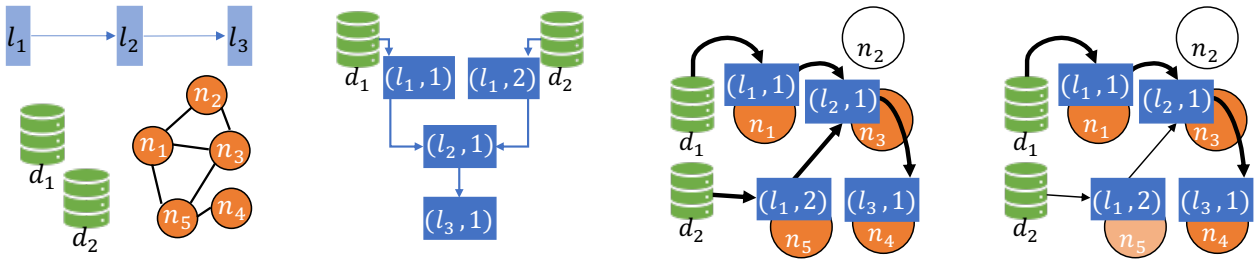


Fig. 1. The different stages of the RightTrain approach. (a): input data, namely, a set of DNN *layers* (light blue), a set of *data sources* (green), and a *physical graph* composed of physical nodes (orange). (b): an instance tree, whose nodes are data sources and *layer instances*. (c): a possible *deployment*, associating layer instances to three out of the five physical nodes. (d): a *refined* deployment, using only some of d_2's data and, accordingly, reducing the computational power allocated to layer instance $(l_1, 2)$.

input data traverses all instances from the leaves to the tree root, and (ii) a *backward pass* where gradients follow the opposite route and local parameters at each layer are adjusted so as to reduce the global loss function. The most commonly-employed optimization algorithm is stochastic gradient descent (SGD), though alternatives tailored to ML have been proposed as well [38].

Thus, the input to our problem includes (see Fig. 1(a)):

- a set $\mathcal{L} = [l_1, \ldots, l_L]$ of *DNN layers* connected to each other to form the DNN structure to implement;
- a set $\mathcal{N}$ of *physical nodes*, i.e., mobile, edge or cloud [4] nodes with the computational capability to run one (or more) layer instances;
- a set $\mathcal{D}$ of *data sources*, which may be colocated with physical nodes.

For each layer $l \in \mathcal{L}$, we know the computational requirement $r(l)$, expressing the amount of computing resources required to process one unit of traffic entering an instance of l (e.g., in CPU cycles per megabit). We are also given coefficients $q(l)$, denoting the ratio between outgoing and incoming data for layer l.

For each node $n \in \mathcal{N}$, we are given the total amount $R(n)$ of available computational[1] resources therein, that can be shared among all layer instances running at n. Parameters $\mu(l,n) \in \{0,1\}$ express whether node n has enough memory[2] to execute an instance of layer l. Concerning data transmission, for each two nodes $n_1, n_2 \in \mathcal{N}$, $S(n_1, n_2)$ indicates the amount of data that can be transferred over the link between them in a time unit, with $S(n_1, n_2) = 0$ denoting nodes out of each other's radio range. Finally, let $\Delta(d)$ be the data generated by source $d \in \mathcal{D}$ at each epoch, and $\eta(d)$ be the node with which d is colocated.

B. Decision variables

The main decisions to make concern (i) how many layer instances to create and how to connect them, as exemplified in Fig. 1(b), (ii) how to deploy the instances onto the physical nodes, as shown in Fig. 1(c), and (iii) how to assign the computational and network resources, as per Fig. 1(d).

Layer instances and instance trees. For each layer $l \in \mathcal{L}$, we shall create at least one and at most $\alpha|\mathcal{D}|$ *layer instances*, with $\alpha \geq 1$ being a redundancy factor. Each layer instance runs at a physical node, and it is identified as a pair (l, i), where i is an index ranging from 1 to $\alpha|\mathcal{D}|$, while the set of instances is denoted by $\mathcal{I}$. As shown in Fig. 1(b), layer instances and data sources in $\mathcal{D}$ are connected to form an *instance tree*, with binary variables $y(l, i, m, j) \in \{0, 1\}$ expressing whether layer instance (l, i) shall be connected to layer instance (m, j). As shown in Fig. 1(b), data sources $d \in \mathcal{D}$ are part of the instance tree, however, each can only be associated with at most one instance $(d, 1)$. Associating zero instances with a certain data source means not using it, e.g., because a sufficient quantity of data is already available.

Deployment and physical graph. Given the set of layer instances and that of physical nodes, we have to decide whether instance $(l, i) \in \mathcal{I}$ shall be deployed at node $n \in \mathcal{N}$; such a decision is expressed through binary variables $z(l, i, n) \in \{0, 1\}$. We will also identify as $\nu(l, i)$ the node at which instance (l, i) is deployed, i.e., such that $y(l, i, \nu(l, i)) = 1$. As for data sources $d \in \mathcal{D}$, values $\nu(d, i)$ identify the physical node data source d is located at.

A further decision concerns the computational resources $\rho(l, i) \leq R(\nu(l, i))$ to be assigned to each instance (l, i) and expressed in CPU cycles per second. Finally, for each data source d, we have to decide the quantity $x(d, 1, m, j) \leq S(\nu(d, 1), \nu(m, j))$ of data to be transferred toward layer instance (m, j). We also indicate with $\chi(l, i, m, j)$ the quantity of data flowing through a generic link from instance (or data source) (l, i) to instance (m, j), defined as:

$$\chi(l,i,m,j) = \begin{cases} x(l,1,m,j) & \text{if } l \in \mathcal{D} \wedge i = 1, \\ q(l) \sum_{(h,k)} \chi(h,k,l,i) & \text{otherwise.} \end{cases} \quad (1)$$

C. Constraints

The decision variables $y(l, i, m, j)$, $z(l, i, n)$, $\rho(l, i)$, and $x(d, 1, l, i)$ are subject to several constraints. Two of them concern the instance tree exemplified in Fig. 1(b) and expressed

[1]Note that a limit on the available energy resources can be included in a straightforward manner.

[2]We take memory as representative of non-computing resources; any other type of resource can be modeled in a similar way.

by the y-variables. Specifically, we must deploy at least one instance of each layer, i.e.,

$$\sum_{i\in[1\dots\alpha|\mathcal{D}|]}\sum_{(m,j)\in\mathcal{I}} y(l,i,m,j)\geq 1,\quad \forall l\in\mathcal{L}. \tag{2}$$

Also, we can only connect on the instance tree *subsequent* layers, i.e.,

$$y(l,i,m,j)\leq \mathbb{1}_{[l \text{ is child of } m]}, \tag{3}$$

Then, moving to the deployment decisions exemplified in Fig. 1(c) and expressed by the z-variables, we must ensure that each instance is deployed at exactly one physical node:

$$\sum_{n\in\mathcal{N}} z(l,i,n)=1,\quad \forall(l,i)\in\mathcal{I}. \tag{4}$$

Last, no layer instance can be deployed at a node lacking the required memory resources:

$$z(l,i,n)\leq \mu(l,n),\quad \forall(l,i)\in\mathcal{I},n\in\mathcal{N}. \tag{5}$$

As for the computational resource allocation and data exchange decisions exemplified in Fig. 1(c) and expressed by ρ and x-variables, we must ensure that the total amount of resources allocated to all the instances running at each node n does not exceed the available one, i.e.,

$$\sum_{(l,i)\in\mathcal{I}:\ \nu(l,i)=n}\rho(l,i)\leq R(n),\quad \forall n\in\mathcal{N}. \tag{6}$$

$$\sum_{\substack{(l,i):\ \nu(l,i)=n\\(m,j):\ \nu(m,j)=n'}}\chi(l,i,m,j)\leq y(l,i,m,j)S(n,n')\quad \forall n,n'\in\mathcal{N}. \tag{7}$$

Finally, we enforce generalized flow conservation [39], i.e., the quantity of data going out of layer instance (l,i) cannot exceed the product between the quantity of incoming data and $q(l)$:

$$\sum_{(m,j)\in\mathcal{I}}\chi(l,i,m,j)\leq q(l)\sum_{(g,h)\in\mathcal{I}}\chi(g,h,l,i)\quad \forall(l,i)\in\mathcal{I}. \tag{8}$$

For data sources, the total quantity of outgoing data cannot exceed $\Delta(d)$:

$$\sum_{(l,i)\in\mathcal{I}}x(d,1,l,i)\leq\Delta(d),\quad \forall d\in\mathcal{D}. \tag{9}$$

D. Objective function

Decisions x, y, z and ρ fully describe the behavior of the distributed learning application. However, they do not directly express: (i) the time taken by each learning epoch, (ii) the energy consumed by each learning epoch, and (iii) the number of epochs needed to attain the required loss function value $\epsilon^{\max}$. We account for these quantities through functions $\mathbf{T}(x,y,z,\rho)$, $\mathbf{E}(x,y,z,\rho)$, and $\mathbf{K}(y,\epsilon)$, respectively. The first two are described in Sec. V-A, while the third one in Sec. V-B.

Given functions $\mathbf{T}$, $\mathbf{E}$, and $\mathbf{K}$, we formulate our problem as minimizing the learning energy consumption, subject to (2)–(9) and to achieving the required loss $\epsilon^{\max}$ by time $T^{\max}$:

$$\min_{x,y,z,\rho}\mathbf{K}(y,\epsilon^{\max})\mathbf{E}(x,y,z,\rho) \tag{10}$$

$$\text{s.t. } (2)-(9),\ \ \mathbf{K}(y,\epsilon^{\max})\mathbf{T}(x,y,z,\rho)\leq T^{\max}. \tag{11}$$

As we will formally prove in Sec. VII, the above problem is NP-hard.

V. Characterizing the Learning Performance

We now show how the performance of the learning process can be characterized, with reference to the time and energy it takes to perform one epoch (Sec. V-A), and the number of epochs necessary to achieve the required learning quality (Sec. V-B).

A. Epoch duration and energy consumption

DNN layers perform linear algebra operations, hence, it is relatively straightforward to characterize the time they take to perform each epoch, and the associated energy consumption. Let us start from epoch time $\mathbf{T}(x,y,z,\rho)$, which depends in a non-trivial manner upon the topology of the instance tree at hand. To compute $\mathbf{T}$, we first define the computational time, $t^{l,i}_{\text{comp}}$, taken by layer instance $(l,i)\in\mathcal{I}$:

$$t^{l,i}_{\text{comp}}=\frac{r(l)}{\rho(l,i)}\sum_{(k,h)\in\mathcal{I}}\chi(k,h,l,i)\,. \tag{12}$$

As per (12), the computation time is given by the ratio between the number of operations to perform (e.g., given by the amount of data to process times the requirement $r(l)$) and the quantity $\rho(l,i)$ of computing resources assigned to that instance. Note that, by constraint (7), the sum in (12) accounts for all the data transferred from the children instances k to the parent instance l. We also define the network time, $t^{n,n'}_{\text{net}}$, needed to transfer data from node n to node n', which depends upon the quantity of data transferred from the children instances k running at n to the parent instance l running at n':

$$t^{n,n'}_{\text{net}}=\frac{\sum_{(h,k),(l,i)\in\mathcal{L}:\ \nu(h,k)=n,\nu(l,i)=n}\chi(h,k,l,i)}{S(n,n')}\,. \tag{13}$$

Then, we can compute times $t^{i,j}_{\text{begin}}$ and $t^{i,j}_{\text{end}}$ at which instance (l,i) starts and ends its computation. Specifically, each instance can only start its processing when all the data it needs from preceding nodes has arrived [18], while its end time is given by the sum between the layer instance begin time and computing time:

$$t^{l,i}_{\text{begin}}=\max_{(h,k)\in\mathcal{I}}\left[y(h,k,l,i)\left(t^{h,k}_{\text{end}}+t^{\nu(h,k),\nu(l,i)}_{\text{net}}\right)\right], \tag{14}$$

$$t^{l,i}_{\text{end}}=t^{l,i}_{\text{begin}}+t^{l,i}_{\text{comp}}\,. \tag{15}$$

Finally, the epoch duration, $\mathbf{T}(x,y,z,\rho)$, is given by the end time of the slowest instance of the last layer, i.e.,

$$\mathbf{T}(x,y,z,\rho)=\max_{(l,i)\in\mathcal{I}:\ l=L}t^{l,i}_{\text{end}}\,. \tag{16}$$

Estimating the energy consumption associated with ML computational tasks has been the focus of a significant body of research [40]. In general, the energy consumption associated with a task (in our case, a learning instance running at a node) is determined by its usage of CPU, memory, and GPU resources [41, Eq. (1)]. Those, in turn, depend upon

the layer implemented, the quantity of data it processes, and the characteristics of the node itself. Thus, for each layer instance $(l,i) \in \mathcal{I}$, we can write:

$$E^{l,i}_{\text{comp}} = t^{l,i}_{\text{comp}} \left[e_{\text{p}}(\nu(l,i))\rho(l,i) + e_{\text{f}}(l,\nu(l,i)) \right]. \quad (17)$$

In (17), we recall that $t^{l,i}_{\text{comp}}$ is the computation time for each layer instance at each epoch. Such a time is multiplied by the power consumed by the node $\nu(l,i)$ at which the instance runs, which depends upon the quantity of resources assigned to it. Parameters $e_{\text{p}}(n)$ and $e_{\text{f}}(l,n)$ express, respectively, the power consumed at node n to provide one unit of CPU, and the power consumed at that node to support the memory and storage requirements of an instance of layer l. Both quantities are parameters for our model and can be set following the methodology introduced in [42], i.e., analyzing the requirements of individual layers and the resulting energy consumption. Furthermore, each instance (l,i) running at node $\nu(l,i) \in \mathcal{N}$ implies additional energy consumption due to data transmissions over the network:

$$E^{l,i}_{\text{net}} = e_{\text{net}}(\nu(l,i)) \sum_{(m,j)\in\mathcal{I}} \chi(l,i,m,j). \quad (18)$$

In (18), the energy spent for data transmissions for layer instance $(l,i) \in \mathcal{I}$ is given by the product of (i) a factor $e_{\text{net}}(\nu(l,i))$, expressing how much energy is required by node n to transmit one unit of data, and (ii) the quantity of data going from (l,i) to any other layer instance (m,j).

The energy consumed during one epoch can be found summing the instance-specific energy consumption values (17):

$$\mathbf{E}(x,y,z,\rho) = \sum_{(i,l)\in\mathcal{I}} \left(E^{l,i}_{\text{comp}} + E^{l,i}_{\text{net}} \right). \quad (19)$$

Finally, $\mathbf{E}(x,y,z,\rho)$ can be mapped into *carbon* emissions following the methodology in [43], which accounts for the quantity of CO_2 emitted for each kilowatt-hour of consumed energy.

B. Overall learning time and energy consumption

In Sec. V-A, we have derived the time $\mathbf{T}$ and energy $\mathbf{E}$ required by each epoch as functions of our decision variables. In order to obtain the total time and energy required by the overall learning process, we need to multiply such values by the number $\mathbf{K}(y,\epsilon^{\max})$ of epochs needed to attain the target learning quality $\epsilon^{\max}$. Deriving a closed-form expression for the number $\mathbf{K}$, in a manner similar to $\mathbf{T}$ and $\mathbf{E}$, is currently possible only for few, simple scenarios among those that we address. Indeed, as discussed in Sec. II, currently-available results only target scenarios where *all* nodes share the same DNN, and *all* layers of said DNN are averaged, i.e., *either* data *or* model parallelism are employed, but not a combination of both. Thus, to show the ability of our approach to deal with arbitrary – and potentially more efficient – instance trees and assignment decisions, we use an auxiliary ML model to estimate $\mathbf{K}$, owing to the ML ability to reveal and leverage the structure underlying those phenomena that are too complex to describe through a traditional model.

A key observation we make is that, if we assume to always use all the available data, i.e., to honor constraints (8) and (9) with the equality sign, then the number $\mathbf{K}$ of epochs to run only depends upon the instance tree we consider, i.e., the $y(l,i,m,j)$ variables. Given such decisions, as well as the target learning quality $\epsilon^{\max}$, our approach follows the steps set forth below:

(i) we run a number M of experiments, each for different values of the y variables;
(ii) for each experiment, we determine the resulting value of $\mathbf{K}(y,\epsilon^{\max})$;
(iii) using the above information, we train an *auxiliary* DNN that can predict the value of $\mathbf{K}(y,\epsilon^{\max})$, given arbitrary values for the y variables.

A similar approach has been used, among others, in [44] for mmWave-based vehicular networks, and in [45] for optical networks.

The output of our experiments is collected as a 5-dimensional tensor whose shape is $|\mathcal{L}|\times\alpha|\mathcal{D}|\times|\mathcal{L}|\times\alpha|\mathcal{D}|\times M$, where $|\mathcal{L}|$ is the number of layers, $\alpha|\mathcal{D}|$ is the maximum number of instances we can create for each layer, and M is the number of experiments we perform. For each experiment $\omega \in \{1\dots M\}$, the corresponding entry in the 5-dimensional tensor contains the decisions $y(l,i,m,j)$. The auxiliary DNN gives as output the number $\mathbf{K}(y,\epsilon^{\max})$ of epochs to run, in order to achieve learning quality $\epsilon^{\max}$.

To identify the best auxiliary DNN, we evaluate three architectures,

1) the basic architecture ("MaxCNN" in plots), with two convolutional layers and two fully-connected ones, and a max-pooling layer after each convolutional one;
2) a variant thereof ("AvgCNN" in plots), where average pooling layers are used *in lieu* of max-pooling ones;
3) a non-convolutional network ("FConly" in plots), where both convolutional layers are replaced by as many fully-connected ones, with the same input and output sizes.

In all cases, we are facing a regression problem, hence, we adopt the mean square error (MSE) as loss function.

We validate our approach using an image classification task leveraging the AlexNet DNN [46] and the CIFAR-10 [47] dataset. The results are summarized in Fig. 2(left), depicting the evolution of the MSE across training epochs. We can observe that, for all the auxiliary DNN architectures, both training and testing MSE values rapidly converge to very small values, below 1 for the FConly architecture. This highlights

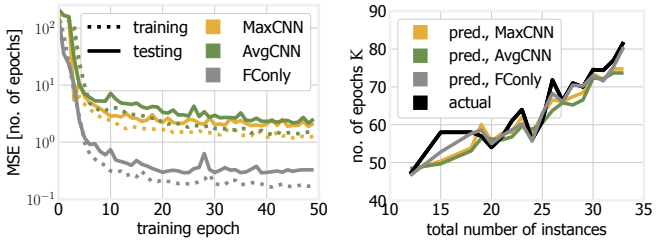


Fig. 2. Auxiliary DNN for estimating the number of required epochs: resulting MSE (left), and real and predicted values of the number K of iterations (right).

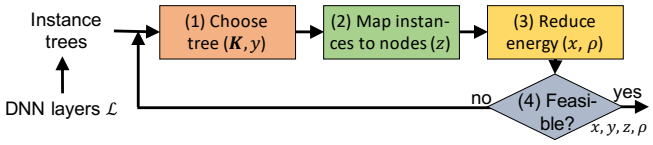


Fig. 3. Main steps of the RightTrain solution concept: given the set of layer instance trees to consider, RightTrain selects, at each iteration, the hitherto-untested tree associated with the lowest energy consumption (Step 1, Sec. VI-A). For such a tree, it makes the near-optimal layer instance-to-node mapping decisions (Step 2, Sec. VI-B), and further improves efficiency by tweaking data and resource utilization (Step 3, Sec. VI-C).

how our approach is indeed very effective in predicting the number K of iterations required for convergence. Fig. 2(right) further shows that, consistently with [18], K tends to grow as the number of layer instances increases, and the FConly architecture is always associated with very close predictions.

Since the applicability of our methodology to real-world cases hinges on the availability of sufficient training data, it is worth highlighting that experiments like ours are routinely performed upon evaluating and adopting a new DNN architecture, hence, obtaining results similar to those in Fig. 2(right) comes at a modest cost in terms of additional work. Moreover, transfer learning could be leveraged to further extend the applicability of the available experiments.

VI. The RightTrain Solution

As proved in Sec. VII, directly optimizing (10) subject to constraints (11), is a daunting task. We thus introduce a new, effective heuristic, called RightTrain, which *decouples* the decisions of (i) choosing the instance tree, (ii) performing instance-to-node mapping, and (iii) assigning the necessary resources. At every step, *efficiency* is the main criterion driving RightTrain decisions.

As summarized in Fig. 3, RightTrain takes as an input the set of instance trees to consider (like the one in Fig. 1(b)); such a set can be efficiently computed offline, in a scenario- and application-dependent manner. RightTrain then iterates over the set of instance trees, selecting at each step the one requiring the least amount of *total* processing (Step 1 in Fig. 3, detailed in Sec. VI-A). For each tree, the y-variables are fixed, hence, in Step 2 (Sec. VI-B) we make the mapping decisions z, under the (temporary) assumption that (i) all the data of the selected sources is used, and (ii) all the processing capabilities at each node are exploited. Both assumptions are dropped in Step 3 (Sec. VI-C), which seeks to refine the solution obtained in Step 2 by using less data and/or less computing power, thereby reducing the energy consumption without jeopardizing the learning performance. If a feasible solution is obtained, then the algorithm terminates (Step 4); otherwise, it goes back to Step 1 and moves to the next instance tree.

A. Layer instance tree ordering

Step 1 of the RightTrain solution requires choosing, from the set of layer instance trees to consider, the next one to try. Ideally, we would like to select a tree minimizing the energy consumption (10), however, this is not possible as

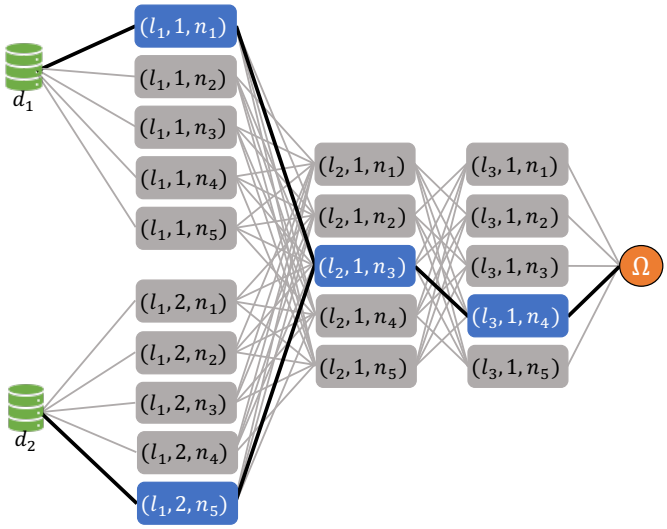


Fig. 4. The expanded graph representing the possible decisions in Fig. 1, with colored nodes and bold edges highlighting the DA-ST corresponding to mapping decisions in Fig. 1(c).

instance-to-node mapping and resource assignment decisions (respectively, Steps 2 and 3) have yet to be made. Nonetheless, the instance tree – along with information on layer and data source characteristics – allows for estimating the total quantity of *processing* entailed by the tree itself. Specifically, recalling that $\mathcal{I}$ is the set of layer instances created for a given tree (i.e., DNN structure) and that y-decisions represent the tree topology, we can express the amount of required processing as:

$$\mathbf{K}(y, \epsilon^{\max}) \sum_{l:\,(l,i)\in\mathcal{I}} r(l) \sum_{\substack{d\in\mathcal{D}:\,(l,i) \text{ is} \\ \text{an ancestor of } d}} \Delta(d) \prod_{\substack{m\in\mathcal{L} \text{ in path} \\ \text{from } d \text{ to } l}} q(m). \qquad (20)$$

Looking at (20) from right to left, the processing required by a given layer of a DNN for each epoch depends upon [17]: (i) the quantity of data it processes (which in turn depends upon the q-coefficients of the layers traversed before l) and (ii) the layer complexity. Such a quantity is then summed across all layer instances, and multiplied by the number $\mathbf{K}(y, \epsilon^{\max})$ of epochs to run before convergence.

In addition to being a sound criterion to follow in all cases, selecting layer instance trees associated with low processing load (20) often results, as proved in Sec. VII, in selecting trees yielding a low value of the objective (10). Indeed, energy consumption (19) is often dominated by the processing energy (17), which in turn depends upon three quantities also accounted for in (20), namely, the number $\mathbf{K}$ of iterations, the layer complexity $r(l)$, and the quantity of available data Δ.

B. Layer instance-to-node mapping

Step 3 of RightTrain *maps* layer instances in $\mathcal{I}$ to nodes in $\mathcal{N}$, i.e., it decides at which node each layer instance should run. As mentioned above, initially such decisions are made under the assumption that all available data and computational capabilities are used. This problem is combinatorial and, in general, hard to approach. On the positive side, however, we

Algorithm 1 Layer instance-to-node mapping

Require: Expanded graph $\{d\} \cup \{(l,i,n)\} \cup \{\Omega\}$
1: $\mathcal{T} \leftarrow \{\Omega\}$
2: **while** $\mathcal{D} \setminus \mathcal{T} \not\equiv \emptyset$ **do**
3: $\quad w^\star, \pi^\star \leftarrow \infty, \emptyset$
4: $\quad$ **for all** $d \in \mathcal{D} \setminus \mathcal{T}, v \in \mathcal{T}$ **do**
5: $\quad\quad w, \pi \leftarrow \mathsf{RestrictedMinWeightPath}(d, v)$
6: $\quad\quad$ **if** $w < w^\star$ **then**
7: $\quad\quad\quad w^\star, \pi^\star \leftarrow w, \pi$
8: $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \pi^\star$
9: **for all** $v = (l,i,n) \in \mathcal{T}$ **do**
10: $\quad z(l,i,n) \leftarrow 1$
11: **return** $\{z\}$

can leverage the *tree* structure connecting layer instances, as in Fig. 1(b). Specifically, our approach is to (i) build the *expanded graph* shown in Fig. 4, summarizing all possible mapping decisions, and (ii) build a delay-aware Steiner tree (DA-ST) upon such a graph. The DA-ST is the minimum-weight tree spanning a given subset of the vertices of an undirected graph, called *terminal vertices*, with the additional constraint that the maximum learning time $T^{\max}$ must be honored, as per the $T^{\max}$-clause in (11).

The expanded graph is built as follows:

1) it contains a vertex (l,i,n) for each possible deployment decision, i.e., for each layer instance-node pair such that $\mu(l,i,n) = 1$;
2) an edge is drawn between vertices (l,i,n) and (m,j,n') if the layer instances are connected in the instance graph and the nodes n and n' can communicate, i.e., if $y(l,i,m,j) = 1 \ \wedge \ S(n,n') > 0$;
3) we also create an additional vertex for each data source in $d \in \mathcal{D}$, and connect such a vertex to all vertices in the expanded graph representing layer instances to which d is connected.
4) we add a further vertex Ω connected to all vertices (l_L,i,n) corresponding to the last layer l_L;
5) the weight of each edge $((l,i,n),(m,j,n'))$ corresponds to the energy consumption, as per (17), due to the deployment decision represented by vertex (m,j,n'), i.e., running instance (m,j) at node n'.

As exemplified in Fig. 4, the DA-ST connects the vertices representing the data sources with Ω, and covers the same topology as the layer instance tree. Since the DA-ST is the *minimum-weight* among the trees with these features, its vertices represent the layer instance-to-node mapping that minimizes energy consumption (see (10)), under the aforementioned conditions, i.e., that all data and computation resources are used. The mapping decisions are made as summarized in Alg. 1. Given the expanded graph whose vertices are (i) data sources in $\mathcal{D}$, (ii) vertices of type (l,i,n) representing possible mappings, and (iii) special vertex Ω (Line 0), we build the DA-ST $\mathcal{T}$. In Line 1, we initialize the tree $\mathcal{T}$, so as to include vertex Ω. Then, so long as there are data sources not yet included in the tree (Line 2), we look for the minimum-weight path such that (i) it connects a data source d not yet in the tree with a vertex v of the DA-ST itself, and (ii) does not break constraints (11). To this end, in Line 3 the minimum weight $w^\star$ and the minimum-weight path $\pi^\star$ are initialized to ∞ and $\emptyset$, respectively. Then function $\mathsf{RestrictedMinWeightPath}(d,v)$ (Line 5) provides the path π connecting each data source d not yet reached by the tree with each vertex v already in the tree. The minimum-weight path is then identified (Line 7) and added to the DA-ST $\mathcal{T}$ in Line 8. Once all data sources have been included, the DA-ST is completed. The algorithm therefore sets to 1 the z-variable corresponding to the selected DA-ST vertices (Line 10), and returns them.

The procedure $\mathsf{RestrictedMinWeightPath}$ uses the algorithm proposed in [48] to find the minimum-weight path connecting v and d while honoring delay requirements. Finding such a path requires solving an instance of the *constrained shortest path* problem. The problem itself is NP-hard, however, the heuristic [48] can solve it within ϵ ($\epsilon \geq 0$) from the optimum in polynomial time.

As proven in Sec. VII, Alg. 1 as a whole has polynomial (namely, $O(|\mathcal{I}|^3 |\mathcal{N}|^3 \frac{1}{\epsilon})$) time complexity has a constant competitive ratio, namely, it is within $\left(2 - \frac{2}{W}\right)(1+\epsilon)$ from the optimum, where W is the number of vertices of the optimal DA-ST.

C. Decisions refinement

In Step 2, we have decided which layer instances to create and how to connect them, i.e., the layer instances in $\mathcal{I}$ and the edges in the layer instance tree connecting them, as expressed through the y-variables, and how to map layer instances to physical nodes, i.e., the z-variables. The values of both variable sets have been obtained under the assumption that all data from the selected data sources and all the capabilities of physical nodes are used. In the spirit of recent efficiency-focused research [17], Step 3 seeks to establish whether *all* that data and that computational power is really needed. Our goal is thus to obtain a solution that meets all constraints in (11), including the minimum learning quality and maximum time, while further improving the energy objective (10).

Given the y- and z-variables, the problem of optimizing (10) subject to constraints (11) only has continuous variables, namely, x and ρ. It follows that such a problem can be efficiently solved: (i) by off-the-shelf solvers, e.g., CPLEX or Gurobi, if a closed-form expression is available for $\mathbf{K}$, $\mathbf{E}$ and $\mathbf{T}$, or (ii) through iterative, gradient-based methods like BFGS [49], if such closed-form expressions are not available. Even more importantly, as formally proven in Sec. VII, such a continuous problem is *convex* in many practical cases. It follows that numerical approaches (either off-the-shelf solvers or gradient-based methods) are *guaranteed* to find the optimal values of x and ρ with very good efficiency – polynomial worst-case complexity [50], and much faster than that in most cases.

VII. Problem and Algorithm Analysis

In this section, we prove several important properties of the problem we solve and the RightTrain approach. We starting from showing that the problem is NP-hard.

Property 1 (Problem hardness)**.** *The problem of optimizing (10) subject to constraints (11) is NP-hard.*

Proof: We prove the thesis through a reduction from the Generalized Assignment Problem [51] (GAP), requiring to assign a set of *tasks* to a set of *agents*; each task-to-agent assignment incurs a given *cost*, and the goal is to minimize the total cost. More specifically, we reduce instances of the GAP to *simpler* instances of our own problem, where:

- there is only one data source $\mathcal{D} = \{d\}$;
- the layer instance graph is a chain, the redundancy factor is $\alpha = 1$ and $q(l) = 1$ for all layers, i.e., there is only one instance per layer and the same traffic traverses them all;
- the number of iterations goes to ∞ if any x-value is lower than $\Delta(d)$, i.e., we must use all data;
- the number of iterations goes to ∞ if any layer instance is assigned less than a quantity ρ_0^l of computational capabilities, and the timeout $T^{\max}$ is set to ∞ – hence, it is optimal to always set $\rho(l,i) = \rho_0^l$;
- communication links have infinite capacity and zero delay.

Due to the conditions listed above, the values of y, x and ρ variables can be trivially set, and our problem reduces to instance-to-node mapping, i.e., setting the z-variables.

Therefore, we can reduce an instance of the GAP problem to an instance of the simplified problem we stated above by:

1) creating one layer (hence, one layer instance) per task;
2) creating one node per agent;
3) setting the fixed energy consumption $e_{\mathrm{f}}(l,n)$ equal to the cost of assigning task l to agent n.

This implies that an instance of a known NP-hard problem, namely, GAP, can be reduced to an instance of ours in polynomial (indeed, linear) time, which proves the thesis. ∎

We remark that the proof of Property 1 reduces GAP instances to greatly simplified instances of our own problem; this allows us to conjecture that our problem, besides being NP-hard, is also significantly more complex than an NP-hard problem like GAP.

Let us now move to the RightTrain heuristic, and focus on Step 1 therein, i.e., the choice of the next layer instance graph to consider. In Sec. VI-A, we formulated a selection criterion based on (20), expressing the quantity of processing associated with a given tree. Next, we prove that such criterion often results in selecting the instance tree with the lowest energy consumption.

Property 2 (Processing and energy)**.** *If proportional energy factors are the same for all usable nodes and dominate the global energy consumption, then a layer instance tree minimizing the quantity of processing (20) also minimizes the objective (10).*

Proof: The objective (10) requires minimizing energy, which is given in (19). Neglecting $e_{\text{net}}(n,n')$ and $e_{\mathrm{f}}(n,l)$, then (19) reduces to $\sum_{(l,i)\in\mathcal{I}} t^{l,i}_{\text{comp}} e_{\mathrm{p}}(\nu(l,i))\rho(l,i)$ which, recalling (12) and considering the conditions stated in Sec. VI-A and that we are not making any mapping decision, the expression becomes $\sum_{(l,i)\in\mathcal{I}} e_{\mathrm{p}}(\nu(l,i))r(l) \sum_{\substack{d\in\mathcal{D}:\ (l,i) \text{ is} \\ \text{an ancestor of } d}} \Delta(d)$. Considering $e_{\mathrm{p}}(n) = e_{\mathrm{p}}\ \forall n$, we can re-write (10) as:

$$e_{\mathrm{p}} \sum_{(l,i)\in\mathcal{I}} r(l) \sum_{\substack{d\in\mathcal{D}:\ (l,i) \text{ is} \\ \text{an ancestor of } d}} \Delta(d) \prod_{\substack{m\in\mathcal{L} \text{ in path} \\ \text{from } d \text{ to } l}} q(m),$$

which is exactly $\frac{e_{\mathrm{p}}}{\mathbf{K}(y,\epsilon^{\max})}$ times the quantity in (20). ∎

We remark that Property 2 describes very well scenarios where layer instances are implemented within, e.g., containers, hence, with very small overhead.

Moving to Step 2 of RightTrain, we show that Alg. 1 has polynomial complexity, and a very good competitive ratio.

Property 3 (Complexity of Alg. 1)**.** *Alg. 1 has a worst-case time complexity of* $O\left(|\mathcal{D}||\mathcal{I}|^3|\mathcal{N}|^3\frac{1}{\epsilon}\right)$.

Proof: As shown in [48], the algorithm implementing the RestrictedMinWeightPath procedure has $O\left(mn(\log\log n\left(1+\frac{1}{\epsilon}\right)\right)$ time complexity, where m and n are, respectively, the number of vertices and edges of the input graph. In our case, the number of vertices of the expanded graph is $O(|\mathcal{I}||\mathcal{N}|)$, and the number of its edges is $O(|\mathcal{I}|^2|\mathcal{N}|^2)$. The whole procedure is repeated at most $|\mathcal{D}|$ times, hence, the thesis follows. ∎

Property 4 (Competitive ratio of Alg. 1)**.** *Alg. 1 has a competitive ratio of* $2(1+\epsilon)$.

Proof: The structure of Alg. 1 mimics that of the algorithm in [52] to solve the Steiner tree problem, which has a competitive ratio of $2 - \frac{2}{W} \leq 2$, where W is the number of vertices in the optimal Steiner tree. However, Alg. 1 contains a further source of suboptimality, namely, the RestrictedMinWeightPath procedure; as per [48], its result is guaranteed to be within $(1+\epsilon)$ from the optimum. Combining the two competitive ratios, the thesis follows. ∎

As for Step 3 of the RightTrain approach, it requires setting the x and ρ variables so as to further improve the energy objective (10), without jeopardizing the constraints in (11). By leveraging theoretical arguments and experimental observations, we can state the following result.

Proposition 1 (Convexity of the problem in Step 3)**.** *The problem of optimizing (10) subject to constraints in (11), with the y- and z-values fixed, is convex.*

The arguments supporting Proposition 1 can be summarized as follows. Constraints (2)–(9) are clearly linear in the variables x and ρ, once y and z are given. Similarly, constraints (13)–(19) reduce to linear expressions in variables x and ρ. Also, (12) is convex in $\rho(l,i)$, as it is easy to verify that its second derivative is always positive. As for the number of iterations needed for convergence, although no closed-form expression for $\mathbf{K}$ is available, theoretical and experimental works [53]–[55] all concur that the relationship between the quantity of used data and the resulting learning quality (e.g., accuracy) is best captured by logarithmic functions, which are convex.

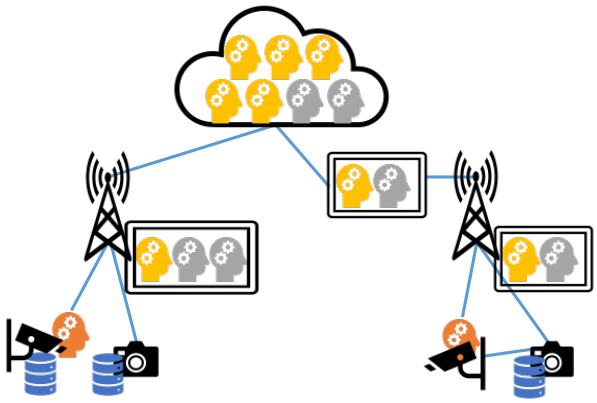


Fig. 5. An example three-layered scenario, including mobile, edge, and cloud nodes. Brains denote different devices with computational capability, with the color of the brain corresponding to the category of the device itself (gold, silver, or bronze). Dark-blue cylinders denote data sources; light-blue edges connect pairs of devices that can communicate.

VIII. Performance Evaluation

After introducing our reference scenarios, in this section we evaluate the performance of RightTrain against split learning (SL) and the optimum (when the scenario size allows it).

A. Reference scenarios

Learning task and DNN. We consider an image classification task over the CIFAR dataset, using a version of the AlexNet DNN [46], including five convolutional layers and three fully-connected ones. Tab. I summarizes the layers composing the AlexNet DNN, along with their complexity (per sample), expressed in millions of operations (MOPs) per sample. Both CIFAR and AlexNet are well-known, widely available and well-studied; this makes our results more significant and easier to reproduce and generalize.

Network scenarios. We consider three-tier scenarios like the one exemplified in Fig. 5, featuring the mobile-edge-cloud continuum and including:

- *user equipment (UE)*, e.g., smart-city devices like cameras and sensors: they may produce data (blue cylinders) and/or have computational capabilities;
- *edge- and cloud-level* datacenters: they contain multiple (virtual) servers.

The computational capabilities of UEs are rated *bronze*, i.e., very limited. Edge- and cloud-level servers, instead, come in *silver* and *gold* variants, the latter with very large, computational capabilities. Importantly, as can be seen from Tab. II, lower-capability servers have better efficiency, i.e., need fewer watts to provide the same number of trillions of operations (TOPs). This suggests that being able to exploit *all* elements of the mobile-edge-cloud continuum, including using less-powerful devices for moderate loads, is an important asset for any decision-making strategy.

We begin our performance evaluation from a *small-scale scenario*, which allows for a comparison again the optimum (see below); this includes four data sources and five computation-capable nodes (three edge servers and two cloud ones). We then move to a *large-scale scenario*, where the number of data sources and nodes grows to 15 and 20, respectively. Further, in the large-scale scenario we introduce a forth type of nodes, denoted as *iron*, with intermediate features between bronze and silver ones.

TABLE I
Complexity of the layers of the AlexNet DNN used for our performance evaluation

Layer name	Type	Complexity [MOPs]
conv1	convolutional	0.043
conv2	convolutional	6.771
conv3	convolutional	10.145
conv4	convolutional	13.523
conv5	convolutional	9.017
fc1	fully-connected	4.001
fc2	fully-connected	16.027
fc3	fully-connected	0.039

TABLE II
Computational capability and power consumption of gold, silver, and bronze servers

Class	Real-world example	Capability [TOPS]	Power consumption [W]	Efficiency [W/TOPs]
Gold	NVIDIA Ampere A100 [56]	312	400	1.28
Silver	NVIDIA RTX A4000 [57]	153.4	140	0.91
Bronze	Apple A14 bionic [58]	11	6	0.54

Benchmark strategies. We compare the performance of RightTrain against SL [14], owing to its power and performance [16]. Specifically, SL splits the DNN into three parts, and aims at running one at each of the mobile, edge, and cloud layers of the network topology; for each layer, the viable server resulting in the lowest energy consumption is chosen. Since we are interested in the *best* decisions that can be made under the SL paradigm, we compare all possible splits, and choose the one resulting in the best value of the objective (10).

Furthermore, as mentioned above, in the small-scale scenario, we compare against optimal decisions, obtained by trying all possible combinations through brute force.

B. Numerical results

The most basic aspect in which we are interested is how effective RightTrain and its counterparts are in pursuing the optimization objective (10). To this end, Fig. 6(left) shows the energy consumed as a function of the maximum learning time $T^{\max}$, for the small-scale scenario. Consistently with intuition, lower values of $T^{\max}$, hence, tighter delay constraints, result in a higher energy consumption.

As for the relative performance of RightTrain and its alternatives, we can identify two distinct regions. When $T^{\max}$ is small, hence, delay constraints are very tight, all strategies perform similarly, with RightTrain consuming slightly less energy than SL and close to the optimum, owing to its greater

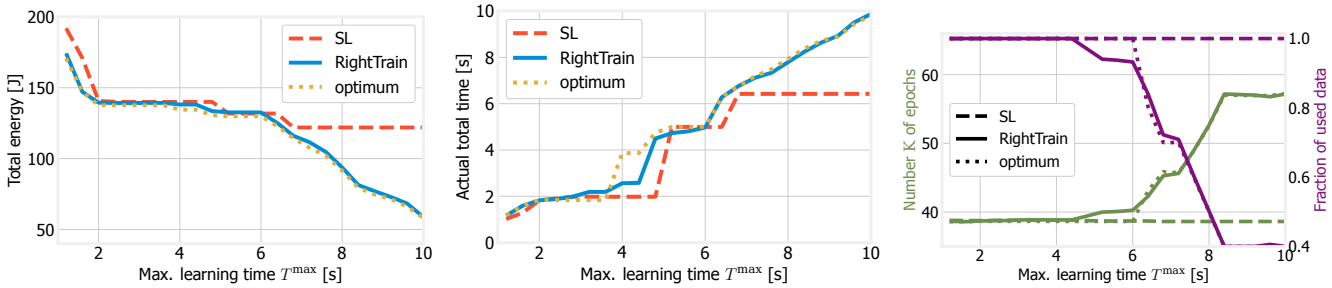


Fig. 6. Small-scale scenario: energy consumed as a function of the maximum learning time $T^{\max}$ (left), actual and maximum learning time (center), number of iterations and fraction of used data (right).

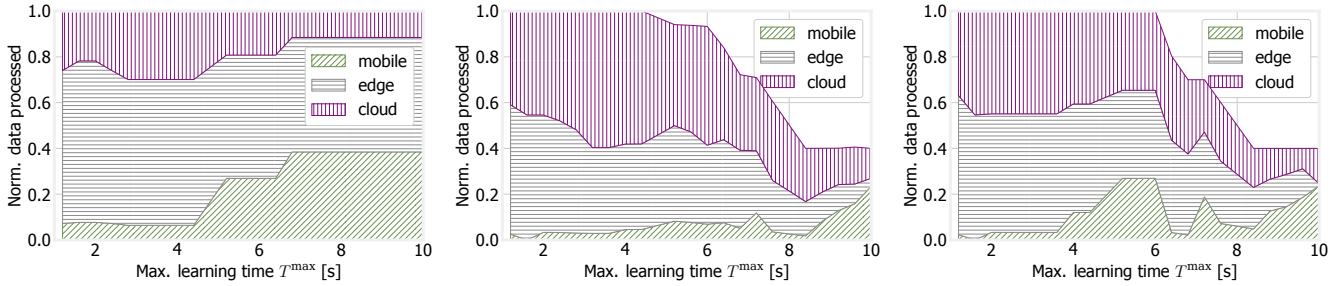


Fig. 7. Small-scale scenario: quantity of data processed at different parts of the network topology as the maximum learning time $T^{\max}$ varies, under the SL (left), RightTrain (center), and optimal (right) strategies.

flexibility in making instance-to-node matching decisions. As $T^{\max}$ increases, we can observe that the energy associated with SL stops decreasing, while RightTrain is able to track the optimum and yield a substantially lower energy consumption, over 50% less than SL. The reason for such behavior is shown in Fig. 6(center): SL can result in learning times that are shorter than $T^{\max}$, especially when $T^{\max}$ itself is higher.

One reason for this is shown in Fig. 6(right), portraying the fraction of data used by each strategy (purple) and the resulting number of iterations K (green). We can see that SL (dashed lines) always uses all available data, which results in a constant (and low) number of iterations. On the other hand, both RightTrain and the optimum are able to use less data when the delay restrictions are looser, achieving a lower energy consumption and, hence, a better efficiency, in spite of a higher number of iterations.

The second reason is shown in Fig. 7, depicting how each strategy utilizes the different parts of the network topology. We can observe that SL (left plot) tends to use more low-powered mobile nodes as $T^{\max}$ increases, as one might expect. For RightTrain (center plot) and the optimum (right plot), the quantity of data to process decreases as $T^{\max}$ increases, which allows for a greater flexibility in using all segments of the mobile-edge-cloud continuum, including high-powered cloud nodes when appropriate. Notice that, under RightTrain and (to an even greater extent) the optimal strategy, the curves in Fig. 7 do not look smooth, e.g., the quantity of data processed at mobile nodes fluctuates as $T^{\max}$ increases. This is in contrast with the monotonic evolution in Fig. 7(left), and reflects the fact that RightTrain is better than SL at accounting for the nonlinearities of the system behavior (e.g., the fixed energy component e_{f}) and it adjusts its decisions accordingly.

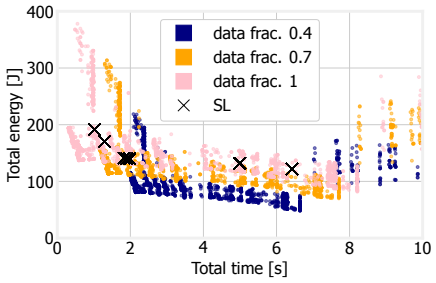


Fig. 8. Small-scale scenario: energy/time trade-offs possible under the RightTrain (dots) and SL (crosses) strategies. Dots of different colors correspond to different fractions of used data.

Fig. 8 provides further insights about the greater flexibility of RightTrain compared to SL. Each marker in the plot corresponds to a possible solution, with its position along the x- and y-axes corresponding, respectively, to its learning time and energy. Dots represent solutions reachable by RightTrain, with their color corresponding to the fraction of used data; black crosses represent solutions reachable by SL. We can immediately see that being able to not use all data allows RightTrain to explore a larger set of high-quality trade-offs, often with a smaller energy consumption and longer learning time. As for SL, all of the solutions it can explore can also be reached by RightTrain when all data is used (pink dots).

We now move to the large-scale scenario and plot, in Fig. 9(left), the energy consumed by the SL and RightTrain strategies (indeed, owing to the scenario size, computing the optimum is not feasible). It is possible to observe a similar behavior to that in Fig. 6(left), with RightTrain always yielding a smaller power consumption than SL, and the difference growing as $T^{\max}$ gets larger. By comparing Fig. 9(left) to Fig. 6(left), it is also possible to observe how RightTrain

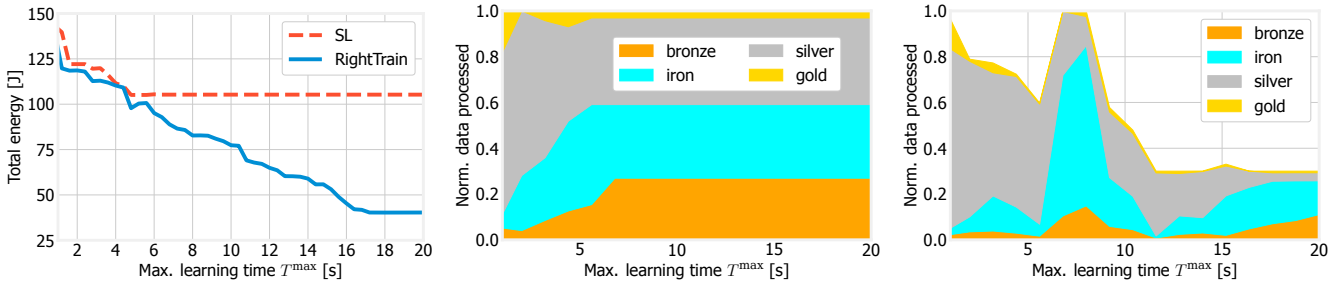


Fig. 9. Large-scale scenario: energy consumed as a function of the maximum learning time $T^{\max}$ (left), and quantity of data processed at nodes of different classes under the SL (center) and RightTrain (right) strategies.

Fig. 10. The nodes of the lab test-bed we employ.

performs noticeably better than SL even for small values of $T^{\max}$, and how the two curves diverge earlier in Fig. 9(left) than in Fig. 6(left).

The reason for such a different behavior is presented in Fig. 9(center) and Fig. 9(right), depicting how (respectively) SL and RightTrain use the different types of nodes in the topology. Similarly to Fig. 7, RightTrain can more flexibly – one would almost say, *creatively* – use available physical nodes, including "iron" ones, thus yielding a lower energy consumption than SL. The behavior and performance difference is more clear here than for the small-scale scenario, due to the wider variety of existing nodes.

IX. Testbed validation

We now validate our model and approach through a lab test-bed composed of three nodes, depicted in Fig. 10:

- a laptop, acting as *edge node*, and equipped with an Intel i7-7700HQ CPU and 8 GB of DDR4 RAM;
- a second laptop, acting as *UE*, equipped with an Intel i7-8550U processor and 16 GB of DDR4 RAM;
- a Raspberry Pi (RPi) 3 Model B, carrying a quad-core 1.2 GHz Broadcom BCM2837 and 1 GB of RAM.

Laptops run the Ubuntu 18.04 operating system, while the RPi runs Ubuntu Server 20.04. UE and edge node are connected through a 3GPP LTE virtualized Radio Access Network (vRAN), leveraging Ettus Universal Software Radio Peripheral (USRP) B210 boards. The vRAN is based on the srsRAN [59] open-source LTE stack implementation, which is compliant with LTE Release 9. The RPi is connected to the UE through Wi-Fi, with the latter acting as an access point.

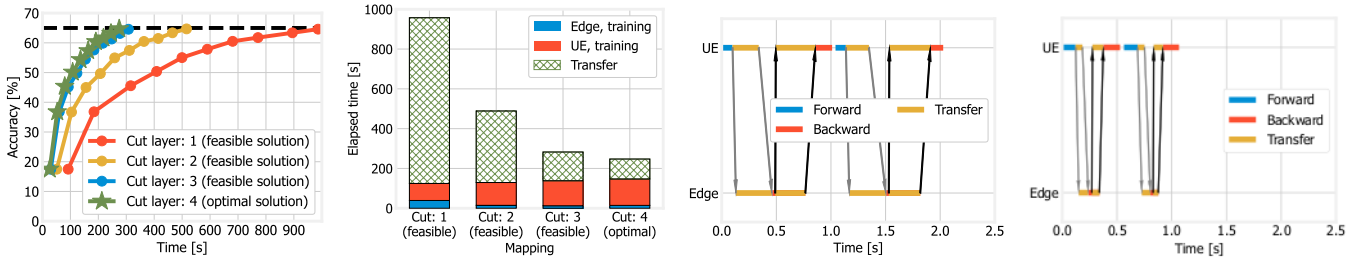
As the learning activity to perform, we consider an image classification task over the CIFAR-10 [47] dataset using the Lenet DNN [60], composed of two convolutional layers followed by three linear ones. We study the performance and behavior of the possible (i.e., feasible) *mappings* between layers and physical nodes, under two test-bed configurations:

- a two-node configuration, where only the Edge node and EU are included and different mapping decisions also imply *cutting* the DNN after a different number of layers;
- a three-node configuration, where the RPi is also used, hence, mapping decisions can be more complex.

In all cases, the target accuracy is set to 65%, and the maximum learning time is 1,000 s.

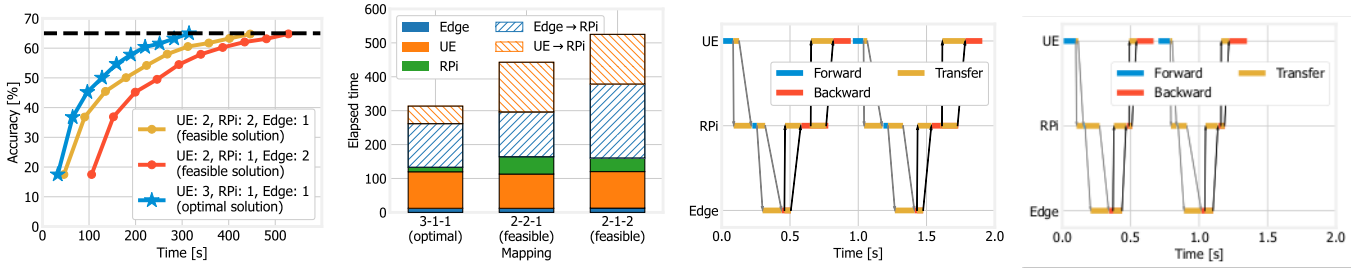
Fig. 11 reports the results for the two-node configuration. From Fig. 11(a), we can observe that 10 epochs are always sufficient to reach the target accuracy; however, the *time* needed to perform such epochs changes significantly; specifically, the later we "cut" the network, the shorter the learning time. The reason of this behavior is highlighted in Fig. 11(b). Interestingly, the total computing time (i.e., considering both the Edge node and the UE) remains roughly constant (since the Edge node and UE laptops have similar performance). On the contrary, the amount of data to be transmitted, hence, the data transfer time, decreases substantially (up to 73%) with higher values of the cut layer, i.e., if we cut the network after a larger number of layers. This is consistent with the fact that later layers of the DNN (i.e., farther from input data) exchange less data, hence, "cutting" the DNN at such layers reduces the quantity of information to exchange between nodes.

Fig. 11(c) and Fig. 11(d) cast additional light on this phenomenon, by depicting how the two nodes alternate performing computations and exchanging data in the first two batches of a typical iteration. For each batch, each node performs the forward step and transmits the output of its own last layer to the following node. Then, when the Edge node terminates the forward stage of the last layer, it computes the loss and starts the backward procedure, computing the gradients and sending them back. The transmissions of the output and gradients are indicated respectively by the gray and black arrows in the plots. Blue and red bars therein correspond



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

Fig. 11. Lab test-bed, two-node configuration: accuracy vs. time for different mappings (a); total elapsed time for different mappings (b); Gantt chart for the "cut layer: 2" (c) and "cut layer: 4" (d) mappings.



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

Fig. 12. Lab test-bed, three-node configuration: accuracy vs. time for different mappings (a); total elapsed time for different mappings (b); Gantt chart for the "UE: 3, RPi: 1, Edge node: 1" (c) and "UE: 2, RPi: 1, Edge node: 2" (d) mappings.

to forward and backward passes, with the latter always taking roughly twice as much as the former. It is easy to notice that, while the forward and backward passes take roughly the same time, "cutting" the DNN at layer 2 (Fig. 11(c)) instead of layer 4 (Fig. 11(d)) results in a much larger quantity of data to transmit, hence, longer total training times.

Fig. 12 confirms the findings above, in spite of the fact that the DNN layers can be spread across three nodes, hence, mappings are more complex. As in Fig. 11(a), in Fig. 12(a) the accuracy reached at each epoch does not change, but the time such epochs take does depend upon the mapping. Importantly, such a time is deeply influenced, as shown in Fig. 12(b), by the transfer times between the nodes.

X. Conclusion and Future Work

We have addressed distributed training of DNNs in the mobile-edge-cloud continuum, and identified the challenge of making joint, energy-efficient decisions on such diverse aspects as (i) selecting the data to be used, (ii) choosing the distributed DNN structure, and (iii) matching DNN layers with the physical nodes to run them. We have presented a solution concept, centered around the RightTrain algorithm, making all necessary decisions in polynomial time and within $2(1+\epsilon)$ from the optimum, with the objective of minimizing the total energy consumption. Our performance evaluation shows that RightTrain closely matches the optimum and reduces the energy consumption of a learning task by over 50% with respect to the state of the art. Furthermore, we have validated our approach by implementing it in a lab test-bed.

Future work will include testing RightTrain against hybrid supervised/unsupervised learning tasks, so as to assess its effectiveness and performance when faced with even larger DNNs and more complex learning techniques. Also, it would interesting to investigate the synergy between RightTrain and model-compression techniques like pruning and knowledge distillation.

References

[1] F. Malandrino, C. F. Chiasserini, and G. Di Giacomo, "Energy-efficient training of distributed dnns in the mobile-edge-cloud continuum," in *IEEE/IFIP WONS*, 2022.

[2] E. Peltonen, M. Bennis, M. Capobianco, M. Debbah, A. Ding, F. Gil-Castiñeira, M. Jurmu, T. Karvonen, M. Kelanti, A. Kliks *et al.*, "6g white paper on edge intelligence," *arXiv preprint arXiv:2004.14850*, 2020.

[3] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[4] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi, "A unified model for the mobile-edge-cloud continuum," *ACM Transactions on Internet Technology*, 2019.

[5] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *ECCV*, 2018.

[6] A. Kappeler, R. D. Morris, A. R. Kamat, N. Rasiwasia, and G. Aggarval, "Combining deep learning and unsupervised clustering to improve scene recognition performance," in *IEEE PIMRC MMSP Workshop*, 2015.

[7] C. Zhuang, A. L. Zhai, and D. Yamins, "Local aggregation for unsupervised learning of visual embeddings," in *IEEE/CVF ICCV*, 2019.

[8] M. Mancini, L. Porzi, S. R. Bulo, B. Caputo, and E. Ricci, "Inferring latent domains for unsupervised deep domain adaptation," *IEEE Transactions on pattern analysis and machine intelligence*, 2019.

[9] M. R. Loghmani, L. Robbiano, M. Planamente, K. Park, B. Caputo, and M. Vincze, "Unsupervised domain adaptation through inter-modal rotation for rgb-d object recognition," *IEEE Robotics and Automation Letters*, 2020.

[10] M. Planamente, C. Plizzari, M. Cannici, M. Ciccone, F. Strada, A. Bottino, M. Matteucci, and B. Caputo, "Da4event: towards bridging the sim-to-real gap for event cameras using domain adaptation," *arXiv preprint arXiv:2103.12768*, 2021.

[11] A. Albaseer, B. S. Ciftler, M. Abdallah, and A. Al-Fuqaha, "Exploiting unlabeled data in smart cities using federated edge learning," in *IEEE IWCMC*, 2020.

[12] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *ACM/IEEE IPSN*, 2016.

[13] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning," *Advances in Neural Information Processing Systems*, 2020.

[14] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[15] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, 2018.

[16] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtepe, H. Kim, and S. Nepal, "End-to-end evaluation of federated learning and split learning for internet of things," *arXiv preprint arXiv:2003.13376*, 2020.

[17] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Communications of the ACM*, 2020.

[18] G. Neglia, G. Calbi, D. Towsley, and G. Vardoyan, "The role of network topology for distributed machine learning," in *IEEE INFOCOM*, 2019.

[19] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, 2019.

[20] M. Merluzzi, P. Di Lorenzo, S. Barbarossa, and V. Frascolla, "Dynamic computation offloading in multi-access edge computing via ultra-reliable and low-latency communications," *IEEE Transactions on Signal and Information Processing over Networks*, 2020.

[21] J. Konečný, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," *arXiv preprint arXiv:1511.03575*, 2015.

[22] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, "Reliable federated learning for mobile networks," *IEEE Wireless Communications*, 2020.

[23] Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, "A learning-based incentive mechanism for federated learning," *IEEE Internet of Things Journal*, 2020.

[24] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Communications Letters*, 2019.

[25] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," in *International Conference on Learning Representations*, 2019.

[26] H. T. Nguyen, V. Sehwag, S. Hosseinalipour, C. G. Brinton, M. Chiang, and H. V. Poor, "Fast-convergent federated learning," *IEEE Journal on Selected Areas in Communications*, 2020.

[27] T. Li, M. Sanjabi, A. Beirami, and V. Smith, "Fair resource allocation in federated learning," in *International Conference on Learning Representations*, 2019.

[28] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, 2019.

[29] 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 Communications*, 2019.

[30] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "A privacy-preserving deep learning approach for face recognition with edge computing," in *USENIX HotEdge*, 2018.

[31] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things," *IEEE Network*, 2019.

[32] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *IEEE INFOCOM*, 2020.

[33] E. Baccour, A. Erbad, A. Mohamed, M. Hamdi, and M. Guizani, "Distprivacy: Privacy-aware distributed deep neural networks in iot surveillance systems," in *IEEE GLOBECOM*, 2020.

[34] T. S. Salem, G. Castellano, G. Neglia, F. Pianese, and A. Araldo, "Towards inference delivery networks: Distributing machine learning with optimality guarantees," 2021.

[35] M. Assran, J. Romoff, N. Ballas, J. Pineau, and M. Rabbat, "Gossip-based actor-learner architectures for deep reinforcement learning."

[36] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge Intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[37] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, 2018.

[38] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *International conference on machine learning*, 2014.

[39] J. Martín-Peréz, F. Malandrino, C.-F. Chiasserini, and C. J. Bernardos, "Okpi: All-kpi network slicing through efficient resource allocation," in *IEEE INFOCOM*, 2020.

[40] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, 2019.

[41] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, "Towards the systematic reporting of the energy and carbon footprints of machine learning," *Journal of Machine Learning Research*, 2020.

[42] X. Mei, X. Chu, H. Liu, Y.-W. Leung, and Z. Li, "Energy efficient real-time task scheduling on cpu-gpu hybrid clusters," in *IEEE INFOCOM*. IEEE, 2017.

[43] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, "Quantifying the carbon emissions of machine learning," *arXiv preprint arXiv:1910.09700*, 2019.

[44] G. H. Sim, S. Klos, A. Asadi, A. Klein, and M. Hollick, "An online context-aware machine learning algorithm for 5g mmwave vehicular communications," *IEEE/ACM Transactions on Networking*, 2018.

[45] C. Rottondi, L. Barletta, A. Giusti, and M. Tornatore, "Machine-learning method for quality of transmission prediction of unestablished lightpaths," *Journal of Optical Communications and Networking*, 2018.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, 2012.

[47] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[48] D. H. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem," *Operations Research Letters*, 2001.

[49] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[50] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*, 2004.

[51] D. G. Cattrysse and L. N. Van Wassenhove, "A survey of algorithms for the generalized assignment problem," *European journal of operational research*, 1992.

[52] H. Takahashi *et al.*, "An approximate solution for the steiner problem in graphs," 1980.

[53] A. A. Abdellatif, C. F. Chiasserini, and F. Malandrino, "Active learning-based classification in automated connected vehicles," in *IEEE INFOCOM PERSIST-IoT Workshop*, 2020.

[54] C. Perlich, F. Provost, and J. S. Simonoff, "Tree induction vs. logistic regression: A learning-curve analysis," *Journal of Machine Learning Research*, 2003.

[55] H. Y. Ong, K. Chavez, and A. Hong, "Distributed deep q-learning," *CoRR*, 2015.

[56] "NVIDIA A100 datasheet," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf, accessed: 2021-07-30.

[57] "NVIDIA RTX A4000 datasheet," https://www.nvidia.com/content/dam/en-zz/Solutions/gtcs21/rtx-a4000/nvidia-rtx-a4000-datasheet.pdf, accessed: 2021-07-30.

[58] "Apple unveils all-new iPad Air with A14 Bionic," https://www.apple.com/newsroom, accessed: 2021-07-30.

[59] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: An open-source platform for lte evolution and experimentation," in *ACM WiNTECH MobiCom Workshop*, 2016.

[60] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.