Tuning DNN Model Compression to Resource and Data Availability in Cooperative Training

Tuning DNN Model Compression to Resource and Data Availability in Cooperative Training

Francesco Malandrino, *Senior Member, IEEE,* Giuseppe Di Giacomo, Armin Karamzade, Marco Levorato, *Senior Member, IEEE,* Carla Fabiana Chiasserini, *Fellow, IEEE*

***Abstract*—Model *compression* is a fundamental tool to execute machine learning (ML) tasks on the diverse set of devices populating current- and next-generation networks, thereby exploiting their resources and data. At the same time, *how much* and *when* to compress ML models are very complex decisions, as they have to jointly account for such aspects as the model being used, the resources (e.g., computational) and local datasets available at each node, as well as network latencies. In this work, we address the multi-dimensional problem of adapting the model compression, data selection, and node allocation decisions to each other: our objective is to perform the DNN training at the minimum energy cost, subject to learning quality and time constraints. To this end, we propose an algorithmic framework called PACT, combining a time-expanded graph representation of the training process, a dynamic programming solution strategy, and a data-driven approach to the estimation of the loss evolution. We prove that PACT's complexity is polynomial, and its decisions can get arbitrarily close to the optimum. Through our numerical evaluation, we further show how PACT can consistently outperform state-of-the-art alternatives and closely matches the optimal energy consumption.**

***Index Terms*—Distributed learning, network support to Machine Learning, model pruning, dynamic programming**

I. Introduction

Training machine learning (ML) models is notoriously hard, as it requires large quantities of data as well as significant computational resources [2], [3]. To cope with this issue, cooperative training – most notably, federated learning (FL) [3]–[5] – has emerged as a nigh-universal approach to leverage the resources of multiple nodes to perform a single learning task. Examples range from smart factory scenarios [6], where model training takes place at both cloud- and edge-based servers, to space applications [7] where models are first trained on the ground and then refined aboard the spacecraft.

In all such scenarios, the data and resources needed to perform the training are scattered throughout different nodes, whose availability and connectivity may significantly vary in both space and time [8]. This results in a major technical challenge, namely, the *mutual adaptation* of the decisions concerning (i) ML training (e.g., model selection and compression), (ii) data selection (i.e., which nodes shall be asked to contribute their datasets for training), and (iii) node and resource allocation (i.e., at which network nodes to train each portion of an ML model). Importantly, the selection of datasets can be performed without sharing the data itself, but based solely on privacy-preserving statistical information [9], [10].

There are three main scenarios where such main mutual adaptation can be beneficial over training a single model (or completely distinct models for different nodes):

- i.i.d. datasets at different nodes – in this case, the main benefit is fine-tuning the resources committed for training to the required learning quality, e.g., by exploiting cheaper nodes;
- datasets that are not i.i.d. but related (e.g., from different domains [6]) – in this case, the same model can be successfully and efficiently trained to work with data from all domains;
- different datasets with convolutional DNNs – as convolutional layers recognize basic features of the data (e.g., simple shapes in images) as opposed to their meaning, their information can be transferred [11], [12] to models using different datasets.

As better discussed in Sec. VII, many existing works address one or another of the aforementioned aspects, but fall short of providing a comprehensive strategy to jointly make all the required decisions. To fill this gap, in this work we focus on deep neural networks (DNNs) and propose a solution strategy and algorithmic framework called Performance-Aware Compression and Training (PACT), supporting all three cases above. PACT creates optimized strategies for the training of DNNs, in presence of (a) heterogeneous nodes, whose datasets cannot be shared, and (b) different DNN models to choose from. A major novelty of PACT is the ability to leverage *multiple* DNN models across different stages of the same learning task, by *switching* among them as needed (e.g., through model pruning [13] or knowledge distillation [14]). For each stage – hence, for each model –, PACT then selects the most appropriate datasets, network nodes, and resources. As in the example depicted in Fig. 1, a fairly complex model may be used in the early stages of training, running on a small set of powerful nodes. Later, it is possible to switch to a simpler (e.g., pruned) model, thus including more nodes with smaller capabilities but more valuable local data [11]. At the same time, the benefits of model switching must be weighed against the cost of switching itself, which requires additional resources and will often result in a (temporary) drop in learning performance.

Our main contributions can be summarized as follows.

(1) Model and problem definition: we develop a comprehensive, synthetic model of networked systems supporting

F. Malandrino and C. F. Chiasserini are with CNR-IEIIT and CNIT, Italy. G. Di Giacomo and C. F. Chiasserini are with Politecnico di Torino, Italy. A. Karamzade and M. Levorato are with the University of California, Irvine, USA.

A preliminary version of this work has appeared at IEEE INFOCOM 2023 [1].

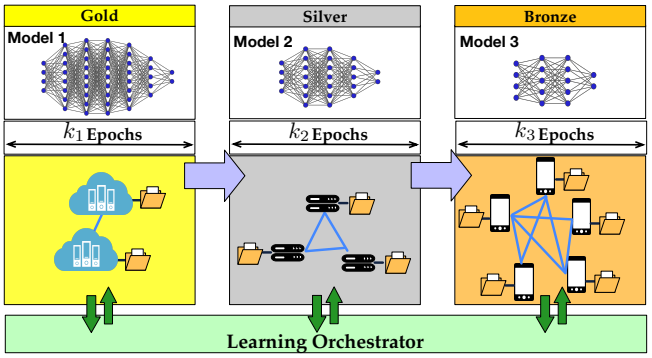

Fig. 1. Cooperative training process proposed and optimized in this paper. Subsets of nodes sequentially train compressed versions of a DNN model. In the picture, nodes are categorized based on their computing capabilities and data availability, and, in the example, the training sequence is based on nodes' ranking (gold, silver, bronze). Our framework, named PACT and running at the learning orchestrator, optimizes the set of nodes, number of epochs, and model compression along the process.

the training of DNN models, capturing all relevant aspects thereof. Leveraging such a model, we formulate the problem of making *dynamic, joint* decisions about: (i) the *models* – including full DNNs or pruned/compressed version thereof – to use at each epoch; (ii) the time and manner of model *switching*, e.g., DNN pruning; (iii) the network *nodes* to use at each epoch, leveraging their computational resources and local datasets. The overall goal is to minimize the energy consumption – hence, cost and carbon footprint – associated with training, subject to constraints about the learning time and quality (e.g., a certain loss function value). Importantly, PACT tackles scenarios where *all* the above aspects can be controlled, thereby achieving greater flexibility and higher-quality decisions than existing works that only target one aspect at a time (e.g., choosing the mode) and consider the others immutable (e.g., resources are given).

(2) Algorithmic framework: making the decisions required in our scenario is complicated by two main issues. The first is common to many combinatorial problems, and is represented by the problem scale, e.g., the vast number of possible solutions to choose from. The second is unique to our own scenario, and is the fact that model switching decisions have effects (most importantly, on the learning quality evolution) that cannot be exactly predicted *a priori*. To tackle the first issue, we adopt an approximate dynamic programming (ADP) approach, predicated on restricting our attention to the most promising potential solutions. Concerning the second issue, we leverage both theoretical works on DNN convergence bounds and data-driven predictions into our solution strategy, so as to estimate the effect of potential decisions with a high level of confidence. As a result, our algorithmic solution can make high-quality decisions – indeed, arbitrarily close to the optimum – with remarkably low (namely, polynomial) computational complexity.

(3) Performance evaluation: we evaluate the performance of PACT under three different real-world scenarios for distributed ML. In all cases, PACT yields training strategies that adapt to the existing resources and training data, honoring the target learning quality and time at a low energy cost. Specifically, PACT decisions are always very close (and, in many cases, identical) to the optimal ones, and substantially better than those made by state-of-the art approaches. We further show how PACT can recover from the effects of inaccurate estimations of the effect of model-switching decisions.

The rest of the paper is organized as follows. Sec. II clarifies the problem we address, while Sec. III presents the system model and the decisions we tackle. Sec. IV then introduces the methodology for estimating the loss as learning proceeds, and Sec. V describes our algorithmic solution. The obtained results are shown in Sec. VI; finally, Sec. VII discusses relevant related work and Sec. VIII summarizes our conclusions.

II. A Motivating Example

In this section, we set in the first of the cases discussed in Sec. I, i.e., i.i.d. datasets, and seek to illustrate the benefits of a cooperative training process that integrates model and nodes switching, but also emphasize the challenges in formulating and optimizing it. To this aim, we consider the case in which one of the most popular cooperative learning approaches, namely, federated learning (FL), is coupled with model pruning [13]. The latter exploits the fact that, typically, many of a model's parameters have a small impact on its performance and can thus be pruned away, resulting in a DNN with similar performance but of lower complexity, and hence CPU and memory requirements. In particular, we evaluate the following scenario:

- the nodes perform an image classification task using the VGG-11 DNN model [15] as a starting point;
- FL uses the cross entropy loss function, batch size equal to 64, and the gradient descent optimizer with 10^{-3} learning rate and 0.9 momentum;
- the model is trained for K_1 epochs on 5 highly capable nodes ("gold" nodes), each using 8,000 randomly-chosen images from the CIFAR-10 dataset [16];
- then, a fraction F of the model's parameters is pruned[1];
- finally, training resumes adding 2 more learning nodes, with lower computing capability and fewer data: either "silver" with half the computing resources of the gold nodes and 2,500 local images each, or "bronze" with one third of the computing resources of the gold nodes and 750 local images each.

Three decisions should be made: (i) the number K_1 of epochs to execute before pruning, (ii) the percentage F of parameters to prune, and (iii) whether to use the "silver" or "bronze" nodes when resuming training. Notice how the first two decisions concern selecting and switching among models, while the third deals with the physical nodes participating in the learning process. Fig. 2 summarizes the effects of such decisions[2], which lead to the following main remarks.

Observation 1: Pruning more (i.e., F=0.9, orange and light blue curves) significantly reduces both CPU consumption (indicated by the numbers in the plot) and epoch duration (markers are closer to each other), thus speeding up the overall

[1]We use the PyTorch method [17] to set to 0 the weights with smaller L^2 norm, and the Simplify library [18] to remove them from the DNN.

[2]Only some values of F are possible, as we apply structured pruning.

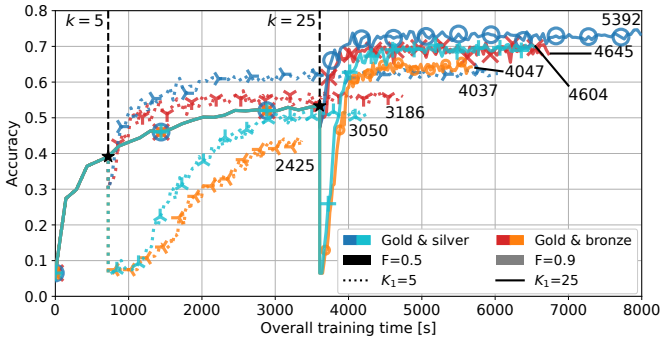

Fig. 2. Accuracy vs. training time for different pruning epoch K_1 (denoted by different line styles) and percentage F (denoted by different color shades). Upon pruning, a sudden drop in accuracy occurs. Cold and warm colors denote the set of nodes used for FL. Numbers indicate the total CPU time [s], while each marker corresponds to 10 epochs.

learning process and reducing its cost.

Observation 2: Larger values of K_1 (solid lines) are associated with better performance after pruning.

Observation 3: Using lower-capability ("bronze") nodes after pruning (warm colors) results in a larger difference between the learning performance obtained when K_1 is small (i.e., 5) and when K_1 is larger (i.e., 25). Thus, achieving better performance while exploiting lower-capability nodes requires switching model later.

In a nutshell, switching from a model to another may have significant benefits in terms of time and resource consumption; however, its effects are hard to capture and foresee. Furthermore, the benefits of involving additional, yet heterogeneous, nodes depend upon the chosen models and the time at which to switch between them. Thus, it is necessary to make all the decisions on model/nodes switching jointly, accounting for their interactions through a comprehensive system model.

III. System Model and Problem Formulation

We now present the system representation and the problem of matching DNN compression and training with resources/data availability.

A. Model components

We envision a networked system for the compression and training of ML models where different nodes or sets of nodes are available, each characterized by computational and energy resources, and local datasets. A *learning orchestrator* controls the learning process. The system has two main components:

- DNN *models* $m \in \mathcal{M}$ that can be used for the training process; each model is obtained by compressing the original DNN with a given technique or pruning ratio;
- sets $n \in \mathcal{N}$ of *nodes* that can participate in the cooperative learning process.

Let k indicate the current epoch, $\ell(k)$ the value of the test loss, computed at epoch k over the orchestrator's dataset, and $T(k)$ the time at which epoch k finishes. $\Delta T(k)$ and $\Delta\ell(k)$ represent, respectively, the time taken by epoch k, and the variation in the value of loss function it yielded. Finally, $\Delta E(k)$ denotes the *energy* consumed to perform epoch k, and $E(k)$ the cumulative energy consumption until k.

Importantly, time and energy variations depend on the model ($m(k)$) and the set of nodes ($n(k)$) used at epoch k; also, they include *two* components each, i.e.,

$$\Delta T(k) = \tau^{\text{change}}(m(k-1), n(k-1), m(k), n(k)) + \tau^{\text{run}}(m(k), n(k)), \quad (1)$$

$$\Delta E(k) = \epsilon^{\text{change}}(m(k-1), n(k-1), m(k), n(k)) + \epsilon^{\text{run}}(m(k), n(k)). \quad (2)$$

In the equations above, τ^{run} (ϵ^{run}) represents the time (energy) to execute a given model over a set of nodes (hence, with the associated datasets), while τ^{change} (ϵ^{change}) represents the time (energy) to change (i.e., *switch*) the model or nodes. In fact, model change implies compressing the model, which may take time and energy, while a change in the set of nodes contributing to learning requires transferring the model. Furthermore, not all model/nodes choices are possible, which is reflected by setting τ^{change}, ϵ^{change}, τ^{run}, and ϵ^{run} to ∞. Also notice how the dependency of τ and ϵ upon the node/cluster n being used allows us to model the fact that the same task takes different time – and result in different energy consumption – if performed at nodes with different architecture and capabilities.

The *evolution* of the loss function is given by:

$$\Delta\ell(k) = \lambda^{\text{change}}(k, m(k-1), m(k)) + \lambda^{\text{run}}(k, m(k), n(k)). \quad (3)$$

Again, (3) includes two components: λ^{change} – the contribution of transitioning from the previous to the current model (if a model switch is performed), and λ^{run} – the effect of training that model for an epoch. The sum of these components gives the difference between the loss at the current epoch k and that of epoch $k-1$, i.e., the result of the action enacted at k.

λ^{run} and λ^{change} describe two different actions with different outcomes: the former corresponds to the usual learning procedure, i.e., running one epoch of training; the latter corresponds to switching across different models, an operation that is done only occasionally and often results in a short-term degradation of the loss. Accordingly, the two components may have different signs: $\lambda^{\text{run}} \leq 0$ (the loss decreases) in most cases, while it is possible that $\lambda^{\text{change}} \geq 0$, as changing model may increase the loss value [19], [20]. Furthermore, the fact that λ^{run} also depends upon the used data allows our model to capture the familiar notion that some datasets are more useful for learning than others. Impossible transitions between learning settings are associated with $\lambda^{\text{change}} = \infty$.

In the following, when no confusion arises, we will drop the dependency of decision variables m and n from the epoch. We present below an example of how our system model can describe concisely and accurately real-world cooperative ML tasks.

Example 1 (System scenario): Consider the scenario in Sec. II. In that case, the set of models $\mathcal{M}$ contains (i) the original model being used, plus (ii) one additional element for

each possible pruning level[3]. For instance, there could be one element for "50% pruning" and another for "90% pruning". Thus, each value of fraction F to prune maps into a different element of $\mathcal{M}$. Concerning number K_1 of epochs to run before pruning, it corresponds to the epoch at which we change the model m being used, hence, $m(K_1) \neq m(K_1+1)$.

Finally, the set $\mathcal{N}$ of possible clusters to use contains three elements: the set of five gold nodes, that of two silver nodes, or the one of two bronze nodes.

In practice, the choice of the elements in $\mathcal{M}$, i.e., the *possible* models to consider, will be done based upon expert – possibly, domain- or scenario-specific – knowledge. A further aspect worth taking into consideration is the availability of information about models, e.g., whether the impact on the learning quality has been profiled as per Sec. IV below.

B. Problem definition

Given the impelling need to make ML sustainable [21], [22], our goal is to minimize the overall learning energy consumption, while ensuring that the loss drops below a target value $\ell^{\max}$ within time $T^{\max}$. Importantly, and unlike many related works, it is not our objective to maximize the learning quality, but rather to minimize the energy consumption subject to learning quality and time targets.

Specifically, for each epoch k, the learning orchestrator has to select (i) which model $m(k)$ to train in epoch k, and (ii) which set $n(k)$ of nodes to involve next in the learning process. Based on these decisions, the values $\Delta T(k)$, $\Delta E(k)$, and $\Delta \ell(k)$ follow, expressing, respectively, how long iteration k takes, how much energy it consumes, and what improvement in the learning it yields. The learning orchestrator is assumed to own a synthetic dataset, which can be either sampled from the participating devices, or obtained through an already trained generative model [23]. By exploiting the testing dataset, it is possible to assess not only the training loss, but also the test loss; using the latter results in better decisions and a lower risk of overfitting. The learning orchestrator acts based on the knowledge of the characteristics of the network nodes that can contribute to a learning process, and of the computational, temporal, and energy impact of running a model. Such values can indeed be calculated following, e.g., the methodology in [24].

The reason why methodologies like [24] are effective is that, contrary to intuition, the operations required by performing one training epoch of a DNN are *deterministic*, e.g., a certain number of matrix products and inversions. Accordingly, given the DNN model to train and the architecture/capabilities of the nodes employ, both the time and energy consumed can be known with virtual certainty. Thus, sets $\mathcal{M}$ and $\mathcal{N}$, as well as functions τ^{run}, τ^{change}, ϵ^{run} and ϵ^{change}, are given from the viewpoint of our problem.

On the contrary, λ^{run} and λ^{change} can only be estimated by the learning orchestrator, through estimators $\hat{\lambda}^{\text{run}}$ and $\hat{\lambda}^{\text{change}}$. This reflects the fact that understanding how training a specific model over specific nodes (hence, also data) improves learning is a hard problem, and, indeed, all existing works merely provide approximations and/or bounds to such quantities. In the following, we treat those estimators as given; then, in Sec. IV we demonstrate one possible methodology that the learning orchestrator can use to compute them.

In general, estimating and modeling the learning performance of DNNs is a distinct, and largely orthogonal, problem to ours; indeed, the overarching level of PACT is to leverage information on DNN performance – regardless of how it is obtained – to efficiently make high-quality learning decisions.

Owing to the discrete-time, combinatorial nature of the problem, we propose an *approximate dynamic programming* (ADP) formulation thereof, as described below. Dynamic programming is indeed well-suited to cope with combinatorial problems where the system state evolves over time and the same decision process shall be repeated for multiple epochs.

C. ADP formulation

First, we define the state space, set of actions, and cost function. The state at epoch k is given by $\mathbf{s}(k)=(k,\ell(k),T(k),m,n)$, while the set of actions available from state $\mathbf{s}(k)$ is given by all possible decisions $(m',n') \in \mathcal{M}\times\mathcal{N}$ such that the switch they entail (if any) is feasible. The *cost function* $\mathbf{C}(\mathbf{s}(k),\mathbf{a}(k))$ expresses the (immediate) cost of executing action $\mathbf{a}$ while in state $\mathbf{s}$ at epoch k, as the corresponding consumed energy $\mathbf{C}(\mathbf{s}(k),\mathbf{a}(k))=\Delta E(k)$. Such a cost comes directly from (2), i.e., $\mathbf{C}(\mathbf{s}(k),\mathbf{a}(k))=\epsilon^{\text{change}}(m,n,m',n')+\epsilon^{\text{run}}(m',n')$.

The *value function* $\mathbf{V}(\mathbf{s}(k))$, i.e., how desirable it is to be in state $\mathbf{s}(k)$, requires a more sophisticated, and domain-specific, definition. We set the value of being in state $\mathbf{s}(k)$ equal to 0 when, after $T^{\max}$, the loss is above $\ell^{\max}$; we set it to the maximum value (i.e., 1) whenever $\ell(k)<\ell^{\max}$ while $T(k)\leq T^{\max}$. For all other states, we compare the current loss $\ell(k)$ and time $T(k)$ with an *ideal* loss-time curve $\ell^{\text{ideal}}(t)$ which: (i) starts at $\ell(0)$ for $T{=}0$; (ii) ends at $\ell^{\max}$ for $T{=}T^{\max}$, and (iii) follows a power law in the between. The latter comes from the finding invariably reported in both theoretical [25]–[27] and experimental [28] works. Then, we can write the value of being in state $\mathbf{s}(k)$ as the difference between ideal and real loss values, i.e.,

$$\mathbf{V}(\mathbf{s}(k))=\text{logistic}\left(\ell^{\text{ideal}}(T(k))-\ell(k)\right), \tag{4}$$

where the value is normalized via a logistic function.

Dynamic programming problems can be solved *in principle* by optimizing Bellman's equation, i.e., choosing at each epoch the action minimizing the total energy cost subject to the constraints that (i) the target quality is reached, i.e., the value of the state reached by the last epoch K is 1, and (ii) such an epoch is performed before the deadline $T^{\max}$ is reached.

$$\min_{\mathbf{a}(k)\in\mathbf{A}^k} \sum_k \mathbf{C}(\mathbf{s}(k),\mathbf{a}(k)) \tag{5}$$

$$\text{s.t. } \mathbf{V}(\mathbf{s}(K))=1 \;;\; T(K)\leq T^{\max}. \tag{6}$$

To solve our problem in real-world scenarios, however, there are two major challenges to face. First, the learning orchestrator does not have access to the future decrease (or increase)

[3]In structured pruning, only a finite set of pruning levels are possible.

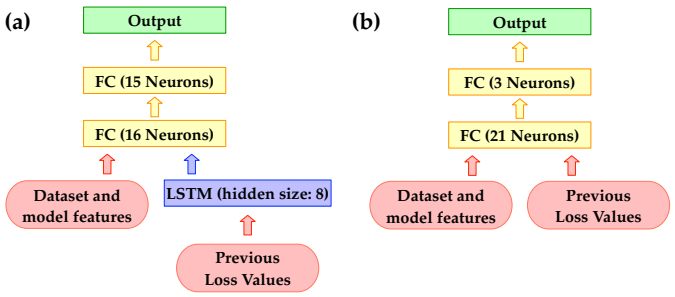

Fig. 3. Architecture of the $\hat{\lambda}^{\text{run}}$ (a) and $\hat{\lambda}^{\text{change}}$ (b) estimators.

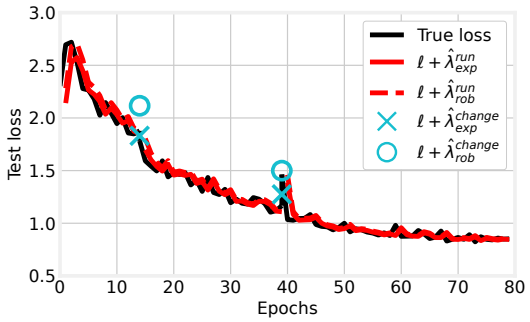

Fig. 4. Example of true loss vs expected-value and robust estimators.

in the loss value $\Delta\ell(k)$, and how our decisions influence it. A possible solution to this issue is to use traditional Deep Reinforcement Learning (DRL) approaches. For instance, Deep Q-Learning algorithms would implicitly learn the probabilistic dynamics of loss as a function of taken actions. However, training DRL agents often requires very large datasets to achieve satisfactory convergence, and may result in weak generalization. Herein, we take a different approach, where we build an ADP framework based on low-complexity neural networks (NN) estimators of possible loss trajectories with a finite time horizon. Second, in view of the number of possible actions, the learning orchestrator has to identify a subset of actions to evaluate at each epoch. Such challenges are dealt with in Sec. IV and Sec. V, respectively.

IV. Estimating the Performance of Learning

As discussed in Sec. III-B, neither of the quantities contributing to the loss evolution ($\lambda^{\text{change}}(k,m,m')$ and $\lambda^{\text{run}}(k,m,n)$) is known exactly. We thus introduce *estimators* for $\Delta\ell(k)$. Specifically, for $\lambda^{\text{run}}(k,m,n)$:

- an *expected-value* estimator $\hat{\lambda}^{\text{run}}_{\text{exp}}(k,m,n)$ of the loss reduction value;
- a *robust* estimator $\hat{\lambda}^{\text{run}}_{\text{rob}}(k,m,n)$, such that $\lambda^{\text{run}}(k,m,n) \leq \hat{\lambda}^{\text{run}}_{\text{rob}}(k,m,n)$ with high probability.

In general, $\hat{\lambda}^{\text{run}}_{\text{exp}}(k,m,n) \leq \hat{\lambda}^{\text{run}}_{\text{rob}}(k,m,n)$, i.e., the robust estimator is the most pessimistic. Likewise, for $\lambda^{\text{change}}(k,m,m')$, we can introduce the corresponding estimators, $\hat{\lambda}^{\text{change}}_{\text{exp}}(k,m,m')$ and $\hat{\lambda}^{\text{change}}_{\text{rob}}(k,m,m')$, with similar properties.

To obtain both the expected-value and the robust estimator, the learning orchestrator leverages the knowledge of the number of classes of the datasets owned by the nodes and makes use of NN architectures that can predict the expected testing loss variation as well as determine the prediction uncertainty. An approach we envision in the following is to estimate $\lambda^{\text{run}}(k,m,n)$ by leveraging the Long Short-Term Memory (LSTM) model in [29] and develop develop a similar, yet simpler, branched architecture, as depicted in Fig. 3(a). Importantly, alternative (possibly, more complex and/or comprehensive) approaches can be equally integrated with PACT.

The features fed to the first Fully Connected (FC) layer are the time-independent parameters, i.e., the number of classes and samples in the dataset of the nodes set currently training the DNN model, and the pruning ratio F of the current model. The input of the LSTM layer is the sequence of loss values obtained so far in the DNN model. The LSTM predicts the expected value of $\lambda^{\text{run}}(k,m,n)$ as well as two associated quantiles (namely, 0.05 and 0.95), yielded by the learning process in the next 5 epochs (thus, the FC layer output size is 15, i.e., number of predicted metrics times number of prediction steps). So doing, we obtain $\hat{\lambda}^{\text{run}}_{\text{exp}}(k,m,n)$ and $\hat{\lambda}^{\text{run}}_{\text{rob}}(k,m,n)$, with the latter given by the 0.95 quantile.

As for $\lambda^{\text{change}}(k,m,m')$, since the goal is to predict the loss variation when we move from one DNN model to another, we leverage regression, using the NN in Fig. 3(b). The NN is fed the pruning ratio and the 5 loss values preceding the model switch. The regression model predicts the expected value $\hat{\lambda}^{\text{change}}_{\text{exp}}(k,m,m')$ as well as the 0.05 and 0.95 quantiles in the next epoch of the DNN training, with $\hat{\lambda}^{\text{change}}_{\text{rob}}(k,m,m')$ being again the 0.95 quantile.

We demonstrate our loss prediction in a small-scale example, as summarized in Fig. 4. We seek to model the testing loss attained by the AlexNet DNN over the CIFAR-10 dataset. The training happens in three steps:

1) the full DNN is trained for 15 epochs at a node containing 16,500 samples of classes 1–6 and 1,000 of classes 7–10;
2) the model is pruned with fraction F_1=0.25, and handed over to a new node owing 12,500 samples (representing all classes equally except 9–10, which are underrepresented) for 25 more training epochs;
3) the model is pruned with F_2=0.5 and handed over to a third node, owing 7,500 uniformly-distributed samples.

In Fig. 4, the true loss is represented by the black line, while the red line and the blue markers represent, respectively, the predicted losses $\hat{\lambda}^{\text{run}}$ and $\hat{\lambda}^{\text{change}}$. It is possible to notice how, even in this relatively small-scale example, the estimators provide remarkably accurate predictions.

Finally, to improve the reliability of the robust estimator, the learning orchestrator compares the values obtained through the above NN to the lower bounds that are available for $\lambda^{\text{run}}(k,m,n)$ [30, Theorem 1] and for $\lambda^{\text{change}}(k,m,m')$ [19, Sec. 3]. If they result to be lower than the bounds, the latter are taken as robust estimators.

V. The PACT Algorithm

The goal of PACT is to let the learning orchestrator efficiently find high-quality solutions to the problem in (5), which,

as shown later, is NP-hard. PACT consists of three steps:

1) Create an *expanded graph* representing the possible decisions and their outcome;
2) Using such a graph, identify a set of decisions deemed *feasible* based on the estimated loss trajectory;
3) By combining learning- and energy-related information, choose the best feasible solution to enact.

Step 1: Expanded graph. The expanded graph is a directed graph built according to the following rules:

- The *vertices* represent the states of the system; they are labeled with the current epoch k, model $m(k)$ and set of nodes $n(k)$ being used, and the total elapsed time $T(k)$ and current loss $\ell(k)$. With the aim of identifying feasible solutions, the latter quantity is computed using the robust estimators $\hat{\lambda}^{\text{change}}_{\text{rob}}(k,m,m')$ and $\hat{\lambda}^{\text{run}}_{\text{rob}}(k,m,n)$;
- Elapsed time and loss values are represented, respectively, with resolutions γ_T and γ_ℓ (e.g., if γ_ℓ=0.1, a vertex with ℓ=0.1 or 0.2 can exist, but not with ℓ=0.15);
- A directed *edge* is drawn between two vertices if there is an action making the system move from one corresponding state to the other; each edge is labeled with the *energy consumption* of the associated action, as in (2);
- Each vertex representing a feasible state of the system (i.e., with $\ell(k) \leq \ell^{\max}$ and $T(k) \leq T^{\max}$) is further connected to a virtual node Ω through a zero-cost edge.

The graph is created through the CREATEEXPANDEDGRAPH function, presented in Alg. 1. First, all *vertices* are created, representing all valid combinations of model and set of nodes, epoch, loss value, and elapsed time (Line 3–Line 6). Note that the quantization parameters γ_ℓ and γ_T (Line 4–Line 6) allow us to control the trade-off between size of the graph and quantization error.

For each vertex v, the effect of taking action $\mathbf{a}$ from vertex v is determined by computing the resulting elapsed time and the required energy (Line 12–Line 13). If either is infinite, then taking action $\mathbf{a}$ while in state v is impossible, and we move on to the next action. Otherwise, the loss ℓ' resulting from taking the action is computed using the robust estimator (Line 16). Now, tuple $(k+1, m', n', \ell', T)$ would describe the state the system lands on after performing $\mathbf{a}$ from v; however, due to the way the vertices are created (i.e., using γ_ℓ and γ_T), such a tuple may not correspond to a vertex in $\mathcal{V}$. Accordingly, in Line 17–Line 18, ℓ' and T' are cast into integer multiples of γ_ℓ and γ_T. Then, vertex v' representing the new state is identified (Line 19), and an edge from v to v' is added using the appropriate energy value E as its weight. Finally, if v is feasible, v is connected to Ω (Line 22).

Fig. 5 presents an example of expanded graph. The initial vertex is associated with epoch k=0, model $m(0)$=m_0, node $n(0)$=n_0, loss $\ell(0)$=1 and elapsed time $T(0)$=0. The learning target is $\ell^{\max}$=0.25 and the time limit is $T^{\max}$=1.5. Also, the resolution values are set to γ_T=0.1 and γ_ℓ=0.1. From the current state, it is possible to change the node (switching to more capable n_1), model (switching quicker-converging m_1), both, or neither; such actions are represented (resp.) by solid green, solid purple, dashed blue, and dotted black edges in the figure. Different combinations of possible switches yield different combinations of loss and elapsed time, only one of which – the bottom, pink vertex – is feasible, hence, connected to Ω.

Algorithm 1 Creating the expanded graph

```
1:  function CreateExpandedGraph
2:      V ← {Ω}                                              ▷ set of vertices
3:      for all m ∈ M, n ∈ N do
4:          for all k ∈ [1, 2, ..., ⌈T^max/γ_T⌉] do
5:              for all ℓ ∈ [0, γ_ℓ, 2γ_ℓ, ..., ℓ(0)] do
6:                  for all T ∈ [0, γ_T, 2γ_T, ..., T^max] do
7:                      v ← (k, m, n, ℓ, T)
8:                      V ← V ∪ {v}
9:      E ← ∅                                                ▷ set of edges
10:     for all v=(k, m, n, ℓ, T)∈V do
11:         for all a=(m', n') ∈ A do
12:             T' ← T+τ^change(m, n, m', n')+τ^run(m, n)
13:             E ← ε^change(m, n, m', n')+ε^run(m', n')
14:             if T'>T^max ∨ E=∞ then
15:                 continue                                 ▷ infeasible, skip this action
16:             ℓ' ← ℓ+λ̂^change_rob(k, m, m')+λ̂^run_rob(k, m, n')
17:             ℓ' ← γ_ℓ ⌈ℓ'/γ_ℓ⌉
18:             T' ← γ_T ⌈T'/γ_T⌉
19:             v' ← (k+1, m', n', ℓ', T')
20:             E ← E ∪ {(v, v', weight=E)}
21:         if ℓ ≤ ℓ^max ∧ T≤T^max then
22:             E ← E ∪ {v, Ω}                               ▷ feasible state
23:     return G=(V, E)
```

Step 2: Feasible paths. Next, PACT uses the expanded graph to identify a set of paths deemed *feasible*; the first edge of such paths represents a feasible action. To mitigate the impact of potential errors in the loss estimation (which in principle may jeopardize feasibility), the expanded graph is built using the *robust* estimators of the loss variation, which guarantees that all paths landing at a feasible node are, indeed, feasible with high probability. Thus, using function FINDFEASIBLEPATHS in Alg. 2, PACT seeks for paths that (i) start from the current state, and (ii) arrive to a feasible state, i.e., to a vertex connected to Ω. Specifically, for each vertex v corresponding to a feasible state, it determines the shortest path (Line 5) from the current state v_{curr} to v. Such paths are collected in set $\mathcal{P}$ and associated with a weight corresponding

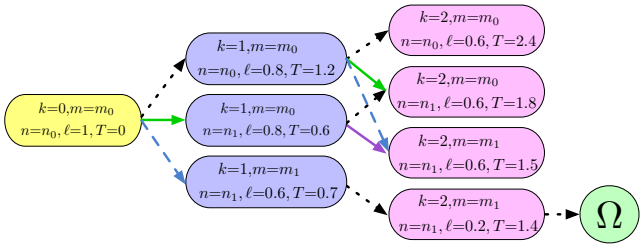

Fig. 5. Example of the PACT expanded graph, with resolution values γ_T=0.1 and γ_ℓ=0.1, learning target $\ell^{\max}$=0.25, and time limit $T^{\max}$=1.5. Edge colors denote switches across subsequent epochs: node only (solid green), model only (solid purple), both (dashed blue), neither (dotted black).

Algorithm 2 Finding feasible paths

1: **function** FINDFEASIBLEPATHS
2: $v_{\text{curr}} \leftarrow (k, m, n, \ell, T)$
3: $\mathcal{P} \leftarrow \emptyset$ ▷ *feasible paths*
4: **for all** v: $(v, \Omega) \in \mathcal{E}$ **do**
5: $p \leftarrow \textbf{shortestPath}(v_{\text{curr}}, v)$
6: $\mathcal{P} \leftarrow \mathcal{P} \cup \{p, \text{weight}{=}\sum_{e \in p} \text{weight}[e]\}$
7: **return** $\mathcal{P}$

Algorithm 3 Choosing the next action

1: **function** CHOOSEACTION
2: $\texttt{scores} \leftarrow \{\}$
3: **for all** $p \in \mathcal{P}$ **do**
4: $\overline{w} \leftarrow 0$ ▷ *opportunity*
5: $\texttt{Le} \leftarrow 0 \quad ; \quad \texttt{Lr} \leftarrow 0$
6: **for all** $((k, m, n, \ell, T), (k', m', n', \ell', T')) \in p$ **do**
7: $\texttt{Le} \leftarrow \texttt{Le} + \hat{\lambda}^{\text{change}}_{\text{exp}}(k, m, m') + \hat{\lambda}^{\text{run}}_{\text{exp}}(k, m, n')$
8: $\texttt{Lr} \leftarrow \texttt{Lr} + \hat{\lambda}^{\text{change}}_{\text{rob}}(k, m, m') + \hat{\lambda}^{\text{run}}_{\text{rob}}(k, m, n')$
9: $\texttt{opp} \leftarrow \texttt{Le}/\texttt{Lr}$
10: $\overline{V} \leftarrow \{v \in \mathcal{V} \colon v[1]{=}m\}$ ▷ *risk*
11: $\texttt{Wr} \leftarrow \min_{\overline{v} \in \overline{V}} \text{weight}(\textbf{shortestPath}(p[1], \Omega, \text{via } \overline{v})$
12: $\texttt{risk} \leftarrow \texttt{Wr}/\text{weight}[p]$
13: $\texttt{scores}[p] \leftarrow \text{weight}[p] \cdot \texttt{risk}/\texttt{opp}$
14: $p^\star \leftarrow \arg\min_{p \in \mathcal{P}} \texttt{score}[p]$
15: **return** $\mathbf{a}{=}(p[1][1], p[1][2])$

to the sum of weights (i.e., energy consumption) of their edges.

Step 3: Making the best decision. Once the set of feasible paths, and associated feasible actions, has been identified, using robust estimators to choose the decision to enact would be overly cautious, possibly resulting in unnecessarily higher energy costs. Thus, PACT accounts for two additional aspects when selecting an action: an *opportunity* and a *risk* factor. Such factors and the path weight are integrated into a *score*, and the action corresponding to the lowest score is enacted.

For every path $p \in \mathcal{P}$, scores are computed in the CHOOSEACTION function in Alg. 3. The opportunity factor, $\texttt{opp} \geq 1$, is given by the ratio of (i) the sum of the expected loss to (ii) the sum of the robust loss associated with the edges in p (Line 9). The intuition is to make it easier to choose actions with a good expected loss, since the robust estimator may be too pessimistic. As for the risk factor, its high-level purpose is to avoid undoing decisions. To this end, PACT seeks for paths on the expanded graph that lead from the first node of p, to a vertex $\overline{v} \in \overline{\mathcal{V}}$ associated with the current model m (Line 10), and thence to Ω. The risk factor, $\texttt{risk} \geq 1$, associated with path p is then computed in Line 12 as the ratio of the minimum among the weights of such paths to the weight of p (defined in Alg. 2).

The score of path p is obtained in Line 13 as p's weight, divided by the opportunity factor, and multiplied by the risk factor. Then the action associated with the minimum-score path is returned. It is important to underline that the shortest path going from the current state to Ω represents the lowest-cost decision since edge weights are set to the energy cost of the corresponding actions. Thus, the ultimate outcome of this step is the action with the lowest energy cost to enact.

A. Problem and algorithm analysis

Property 1: The problem of optimizing (5) is NP-hard.

The proof is based on a reduction in polynomial time from the generalized assignment problem (GAP) [31], which is known to be NP-hard. Furthermore, we prove that:

Property 2: PACT's time complexity is polynomial.

Proof: PACT's complexity is given by the sum of the complexity of Alg. 1–Alg. 3. In Alg. 1, the first loop is run at most $|\mathcal{V}| = MN \left\lceil \frac{T^{\max}}{\gamma_T} \right\rceil^2 \left\lceil \frac{\ell(0)}{\gamma_\ell} \right\rceil$ times, and the second one for at most $|\mathcal{V}|MN$ times. Alg. 2 computes at most $|\mathcal{V}|^2$ shortest paths, each of which (e.g., using Dijkstra's algorithm [32]) incurs a polynomial complexity. Alg. 3 iterates over set $\mathcal{P}$ of feasible paths, whose number cannot exceed $|\mathcal{V}|$ (as per Alg. 2, Line 4). Thus, Alg. 1 represents the dominating contribution to PACT's complexity, which proves the thesis. ■

Importantly, Property 2 concerns the *worst-case* time complexity of PACT, which in practice has substantially lower complexity. In particular, the shortest-path routines used in Alg. 2 and Alg. 3 have been heavily optimized, and perform very efficiently in practice [32].

We can further prove that the space complexity of PACT's most complicated part, i.e., the CREATEEXPANDEDGRAPH procedure in Alg. 1, does not exceed that of its output, i.e., the expanded graph itself.

Property 3: The space complexity of Alg. 1 is $|\mathcal{V}|^2 MN$, with $|\mathcal{V}| = MN \left\lceil \frac{T^{\max}}{\gamma_T} \right\rceil^2 \left\lceil \frac{\ell(0)}{\gamma_\ell} \right\rceil$

Proof: The proof follows by inspection of Alg. 1. First, we remark that no data structures are created within the algorithm. Then we observe that at most one vertex is created every time the first loop in the algorithm is ran, and at most one edge is created every time the second loop is ran. Considering (see also the proof in Property 2) that the first loop runs at most $|\mathcal{V}| = MN \left\lceil \frac{T^{\max}}{\gamma_T} \right\rceil^2 \left\lceil \frac{\ell(0)}{\gamma_\ell} \right\rceil$ times, and the outer loop at most $|\mathcal{V}|MN$ times, then the thesis holds. ■

The intuitive meaning of Property 3 is that the space complexity of Alg. 1 does not exceed that of its output, further supporting the suitability of PACT even to large-scale, complex scenarios.

At last, we prove the following property about how good PACT's solutions are at minimizing the objective in (5).

Property 4: If predictions are exact, their time horizon is sufficiently long, and all $\Delta\ell$ and ΔT values are integer multipliers of γ_ℓ and γ_T, then PACT is optimal.

Proof: The proof comes from inspection of Alg. 1–Alg. 3, which consider all possible options, hence, no feasible solutions are ignored. Further, the shortest-path problem in Alg. 2 and Alg. 3 can be efficiently solved to the optimum. If the hypothesis holds, then the ceiling operators in Alg. 1 (Line 17 and Line 18) have no effect, hence, there is no possible source of suboptimality. ■

An important consequence of Property 4 is that, by varying γ_ℓ and γ_T, we can effectively trade off how close to the optimum the solution gets with PACT's time complexity.

TABLE I
EXAMPLE ACTIONS AND THEIR EFFECTS IN THE SEQUENTIAL LEARNING AND FEDERATED LEARNING SCENARIOS

epoch	model	node/cluster	energy	learning
Sequential Learning				
30	B	Silver	315	1.09
31	B	Silver	321	1.09
40	B	Silver	375	1.08
31	D	Bronze	317	1.22
40	D	Bronze	335	0.90
Federated Learning				
16	B	Silver	15.5	1.56
22	B	Silver	18.5	1.35
22	C	Bronze	17.9	1.27

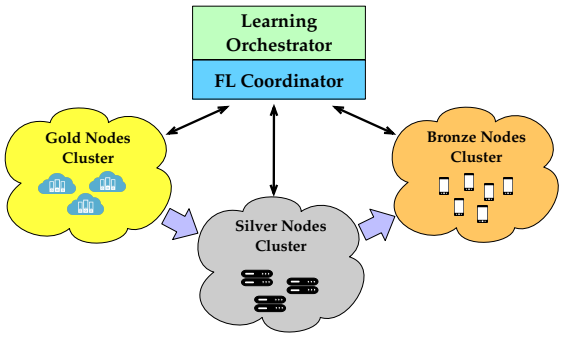

Fig. 6. Federated scenario. Nodes of the same category form a cluster, within which learning is performed in parallel, employing FedAvg; the aggregation of the models is then performed by an FL coordinator. Over the clusters, instead, the learning procedure takes place sequentially.

VI. NUMERICAL RESULTS

We assess PACT's performance focusing on a smart factory-based application under two different learning scenarios:

- a *sequential learning* scenario, like the one depicted in Fig. 1, where models are passed among individual nodes;
- a *federated learning* scenario, as depicted in Fig. 6, where *clusters* of nodes are employed instead.

In both scenarios, we perform an image classification task, using the CIFAR-10 dataset.

Importantly, the PACT methodology – e.g., building the expanded graph and finding a shortest path therein – works unmodified in both scenarios. Needless to say, the available actions (i.e., node-selection and model-switching decisions), as well as their effects on the learning quality and energy consumption, are different and are estimated as discussed in Sec. IV. An example of possible decisions and their effect is presented in Tab. I.

Sequential scenario. We consider three nodes, each belonging to a different category, namely, gold, silver or bronze. They have respectively 17,500, 12,500, and 7,500 samples from the CIFAR-10 dataset. While the bronze node has a balanced data distribution, the gold and the silver ones have unbalanced datasets: the gold node has 2,750 for each of classes 1–6 and 250 for each of classes 7–10; the silver node has 1,500 samples for each of classes 1–8 and 250 for each of classes 9–10; finally, the bronze node has 750 samples per class. In this situation, the most capable nodes do not necessarily possess the highest-quality datasets, hence, trivial decisions (e.g., always using the gold node) are unlikely to yield good performance. Therefore, learning optimization strategies like PACT becomes necessary.

TABLE II
LOSS PREDICTION: MEAN AND STANDARD DEVIATION FOR ALL SCENARIOS

Scenario	Model	MAE	MIL	ICP
Sequential VGG	$\hat{\lambda}^{\text{change}}$	0.222 ± 0.006	0.90 ± 0.04	0.89 ± 0.02
	$\hat{\lambda}^{\text{run}}$	0.0123 ± 0.0002	0.0567 ± 0.002	0.89 ± 0.01
Sequential AlexNet	$\hat{\lambda}^{\text{change}}$	0.077 ± 0.008	0.38 ± 0.03	0.92 ± 0.02
	$\hat{\lambda}^{\text{run}}$	0.0092 ± 0.0006	0.038 ± 0.003	0.89 ± 0.01
Federated VGG	$\hat{\lambda}^{\text{change}}$	0.0270 ± 0.004	1.07 ± 0.02	0.895 ± 0.009
	$\hat{\lambda}^{\text{run}}$	0.010 ± 0.001	0.036 ± 0.004	0.87 ± 0.05

Learning always starts with the gold node training the full model. Then either one or two pruning steps (i.e., two or three models) are considered, with pruning being performed as described in Sec. II. In the first case, after K_1 epochs, the model is pruned with pruning ratio F_1 and handed over to the silver or the bronze node, which continues the training. If instead two pruning steps are performed, then the training at the silver node is interrupted after K_2 epochs, after which the model is pruned with fraction F_2 and sent to the bronze node. Two convolutional DNNs are considered, namely, VGG-19 and AlexNet [33].

Importantly, different combinations of F_1 and F_2 correspond to different elements of the models set $\mathcal{M}$, hence, setting those values is equivalent to selecting the models to use. Recall that model pruning also implies switching to a different node, so that more complex models are always matched with more capable nodes. Specifically, the training time and energy values used for the gold, silver, and bronze nodes reflect (resp.) the capabilities of NVIDIA Ampere A100 [34], NVIDIA RTX A4000 [35], and Raspberry Pi's Videocore 6 [36] GPUs. Finally, for simplicity, we set a very long time limit of $T^{\max}=1,000$ time units. Notice how the energy cost associated with nodes is incurred only *while* the nodes themselves are used, i.e., the energy consumption of idle nodes is neglected [21], [22].

Federated scenario. In the FL scenario, we replace individual nodes with *clusters*, each including two nodes and a learning coordinator. The latter performs model averaging after each epoch following the FedAvg algorithm [37], on the grounds that it is the vanilla approach to FL, hence, provides the easiest-to-replicate results. Note that the FL coordinator (running FedAvg) and the learning orchestrator (running PACT) are two different logical roles, which may not necesarily be taken on by the same physical node.

A. Loss prediction implementation

As described in Sec. IV, we use a DNN and an LSTM to estimate (resp.) λ^{run} and λ^{change}. To capture the difference between different settings, we train separate models for each of our scenarios; further, in the sequential scenario, we train separate models for the cases when the VGG and AlexNet

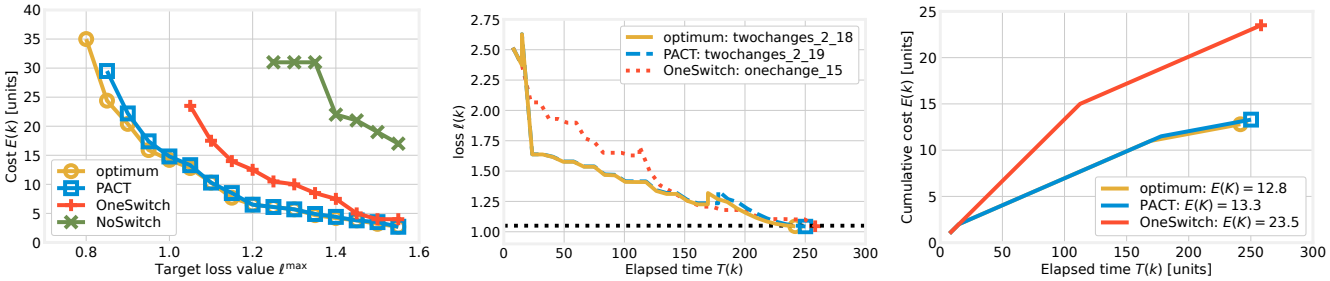

Fig. 7. PACT and benchmark strategies for sequential scenario with VGG: cost for different values of $\ell^{\max}$ (left); evolution of loss (center) and cost (right) when $\ell^{\max}$=1.05.

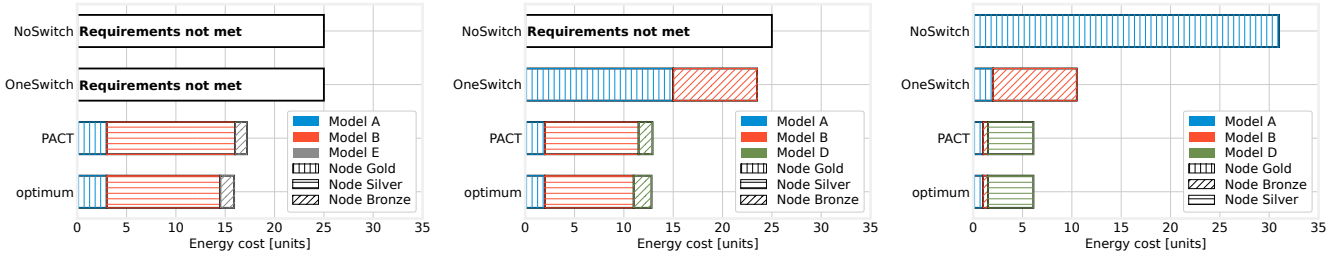

Fig. 8. PACT vs. benchmark strategies for sequential scenario with VGG: energy cost incurred by using different models when $\ell^{\max}$=0.95 (left), $\ell^{\max}$=1.05 (center), $\ell^{\max}$=1.25 (right).

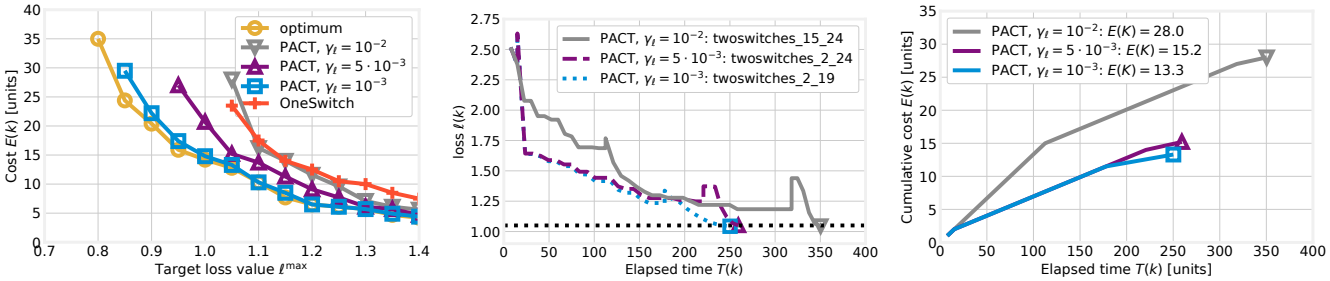

Fig. 9. Impact of γ_ℓ on PACT's performance for sequential scenario with VGG: energy cost for different values of $\ell^{\max}$ (left); evolution of loss (center) and cost (right) when $\ell^{\max}$=1.05.

networks are used. Because we are interested in estimating the whole distribution of the loss, we employ a customized loss function given by the summation of the mean square error (MES) and a *tilted loss term* [38] ensuring that all quantiles are correctly estimated.

The prediction quality achieved is reported in Tab. II, summarizing the prediction metrics we consider, namely: (i) the mean absolute error (MAE); (ii) the mean interval length (MIE), i.e., the average width of the prediction interval; (iii) the interval coverage percentage (ICP), i.e., the fraction of true values falling within the relative prediction interval. The latter two metrics are linked with the quality of quantile predictions. From the table, it clearly emerges that the loss prediction is very good in all scenarios, a further evidence of the validity of the PACT solution strategy and the underlying intuition.

B. PACT performance

Benchmark solutions. We compare the performance of PACT against the following benchmarks: (i) *Optimum*: the optimal decisions yielding the minimum cost, found through brute-force search and using the true loss evolution; (ii) *NoSwitch*: no model switching occurs, meaning that only the full model is used; (iii) *OneSwitch*: only two models are used. For both the *NoSwitch* and the *OneSwitch* solution, we consider the best decisions they yield for each value of $\ell^{\max}$. Specifically, for *OneSwitch*, we consider the lowest energy cost, feasible strategy changing once, considering all combinations of models and changing epochs. Note that most state-of-the-art works [14], [39], [40] envision pruning once, hence, their performance would be represented by *OneSwitch*.

Sequential scenario, VGG model. First, we evaluate PACT's effectiveness, i.e., how the cost (i.e., the consumed energy $E(K)$) it yields compares to that of the benchmarks. To this end, Fig. 7(left) shows the cost associated with each strategy, for different loss targets $\ell^{\max}$. We can observe that the NoSwitch strategy is the worst one and does not allow reaching a low value of $\ell^{\max}$, as only the full model is used. Recall that the full model is trained by the gold node, which has many data samples, but distributed in an extremely unbalanced manner, as 4 out of 10 classes are highly under-represented. Excluding the NoSwitch strategy, when $\ell^{\max}$ is relatively high, all strategies result in very similar performance; on the other hand, they diverge as $\ell^{\max}$ decreases, i.e., as the conditions become more challenging. In particular, PACT closely matches

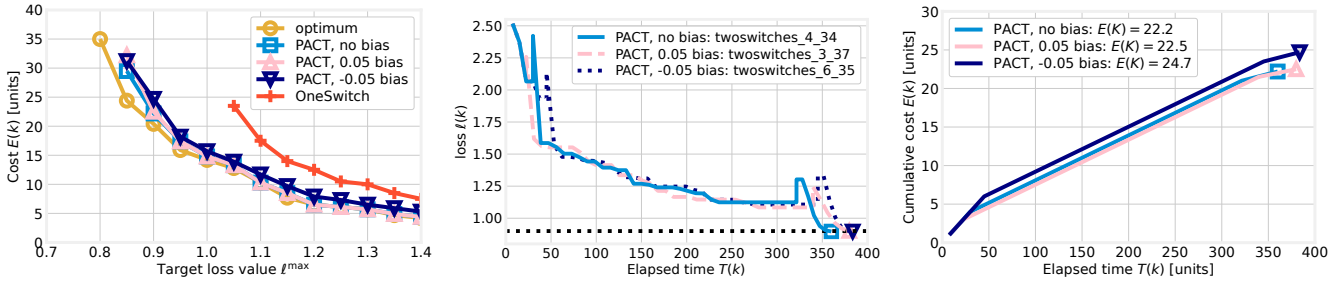

Fig. 10. Impact of quality of $\hat{\lambda}^{\text{run}}$ for the sequential scenario with VGG: PACT's energy cost for different values of $\ell^{\max}$ (left); loss (center) and cost (right) evolution for $\ell^{\max}$=0.9.

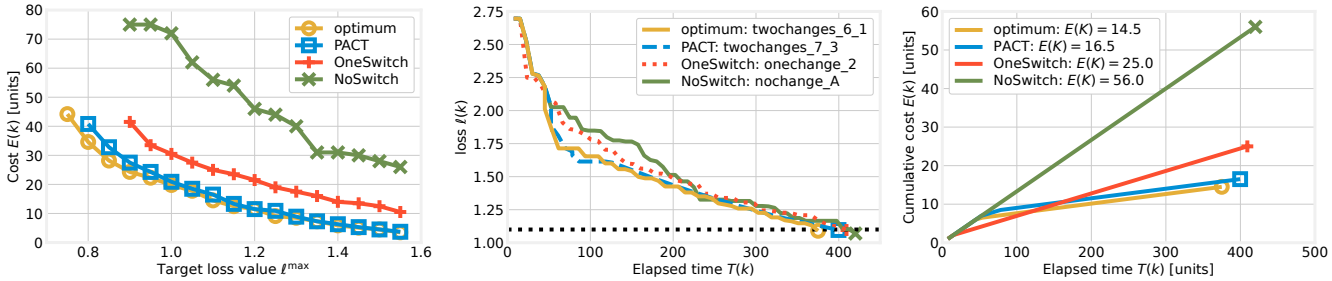

Fig. 11. PACT and benchmark strategies for sequential scenario with AlexNet: cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max}$=1.1.

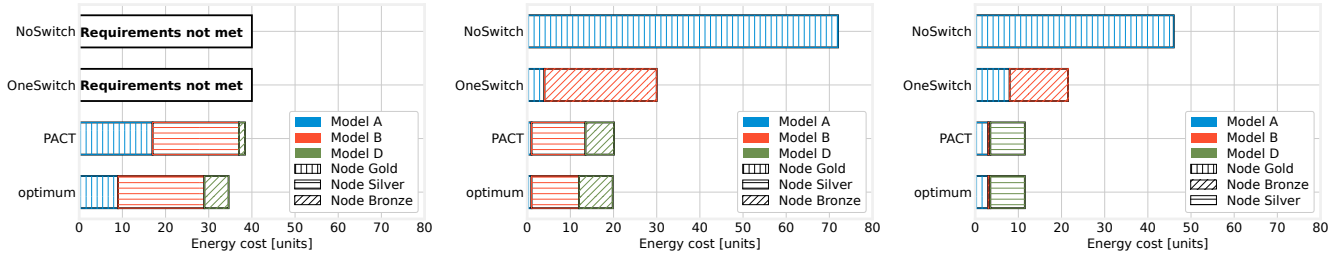

Fig. 12. PACT vs. benchmark strategies for sequential scenario with AlexNet: energy cost incurred by using different models when $\ell^{\max}$=0.8 (left), $\ell^{\max}$=1 (center), $\ell^{\max}$=1.2 (right).

the optimum, to the point that the corresponding curves almost overlap, and outperforms OneSwitch solution – which is the approach followed in most state-of-the-art works. Switching across *multiple* models and nodes is indeed beneficial when learning constraints are tight.

Fig. 7(center) depicts the time evolution of $\ell(k)$ for $\ell^{\max}$=1.05. Note that the peak due to the loss variation λ^{change} incurred upon model switching is not always present, as the testing loss can decrease even when switching the model. This is especially true when the first switching occurs early, as depicted by the two curves with K_1=2, relative to the PACT and the optimal solutions. Remarkably, PACT makes virtually the same decisions as the optimal policy, i.e., performs the model switching at (almost) the same times. OneSwitch can only switch once, hence, does so later. Interestingly, all the strategies achieve the learning target at almost the same time. However, we recall that *cost* is the optimization objective (5), while time is a mere constraint. Accordingly, Fig. 7(right) highlights how the optimum indeed takes slightly shorter than PACT to reach the objective and does so at a (marginally) lower cost (see the position of the last marker on the y-axis). On the other hand, OneSwitch solution, despite taking a comparable time, requires much more energy to reach $\ell^{\max}$.

Fig. 8 sheds further light on how different strategies use the network infrastructure. Plots therein show how much energy is spent running each of the distinct models under the different strategies; different plots correspond to different values of $\ell^{\max}$, that are 0.95, 1.05, and 1.25 (resp.). Consistently with Fig. 7(right), when $\ell^{\max}$ is high, NoSwitch requires the larger amount of energy, i.e., higher cost, while PACT and optimum incur almost the same cost. As for OneSwitch, the energy cost is between the one of PACT and NoSwitch. As shown in Fig. 8(center), for a medium value of $\ell^{\max}$, NoSwitch cannot achieve the target loss value. Thus, the OneSwitch solution requires the highest amount of energy, while PACT and optimum decisions entail a very similar cost, even if the decisions are a bit different, as depicted in Fig. 8(center) by the different costs of models B and D. When $\ell^{\max}$ is low, as in Fig. 8(left), the difference between PACT and OneSwitch emerges more clearly: PACT requires more energy and takes different decisions, employing model B more. Further, in this case, also OneSwitch solution cannot achieve the desired $\ell^{\max}$.

Next, we assess the impact of γ_ℓ and γ_T, which control the trade-off between PACT's complexity and representation's granularity. Fig. 9(left) shows that a larger value of γ_ℓ does

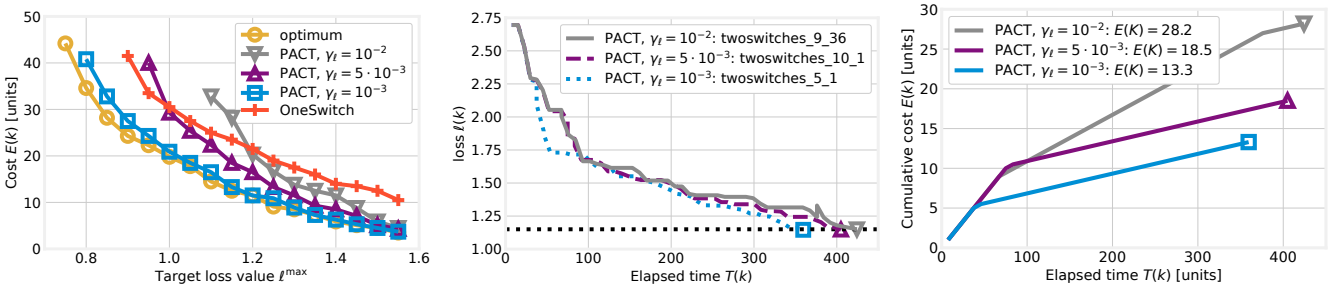

Fig. 13. Impact of γ_ℓ on PACT's performance for sequential scenario with AlexNet: energy cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max}$=1.15.

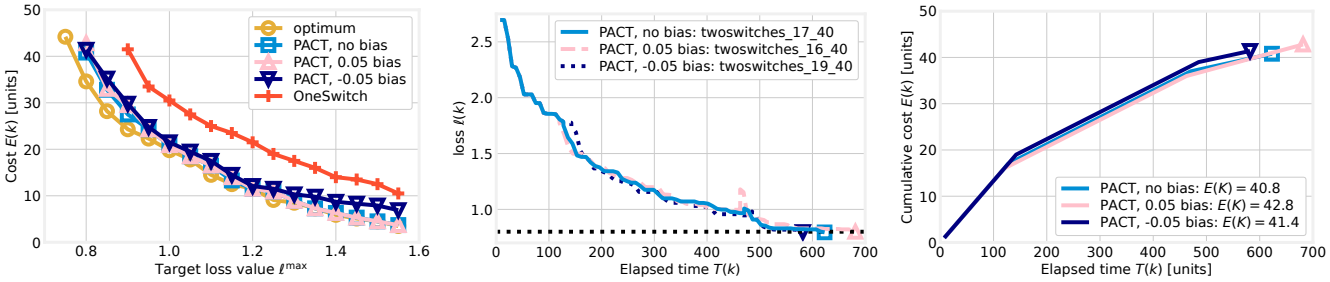

Fig. 14. Impact of quality of $\hat{\lambda}^{\text{run}}$ for sequential scenario with AlexNet: PACT's energy cost for different values of $\ell^{\max}$ (left); loss (center) and cost (right) evolution for $\ell^{\max}$=0.8.

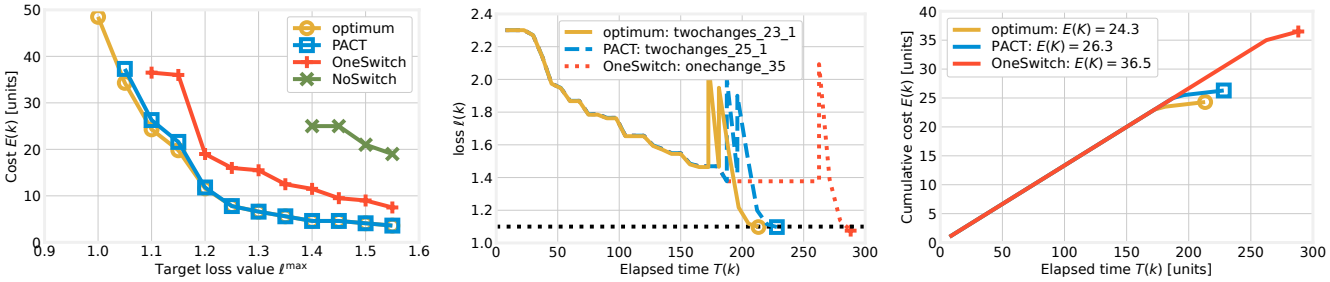

Fig. 15. PACT and benchmark strategies for the federated scenario with VGG: cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max}$=1.1.

decrease PACT's performance: indeed, the minimum value of $\ell^{\max}$ that can be achieved significantly increases with γ_ℓ. However, even increasing γ_ℓ by an order of magnitude, PACT still outperforms OneSwitch for large $\ell^{\max}$, while with lower $\ell^{\max}$ it is comparable to, or slightly worse than, OneSwitch.

Fig. 9(center), referring to the case $\ell^{\max}$=1.05, provides some insight on *how* a higher γ_ℓ affects the decisions made by PACT. Specifically, the higher the value of γ_ℓ, the later switches are made. The reason lies in Line 17 of Alg. 1, and more exactly in the ceiling operator therein. Increasing γ_ℓ leads to overestimating the loss resulting from a particular action, hence, to assume that further gains could be made under the current model, while that is not the case. For the same value of $\ell^{\max}$, Fig. 9(right) highlights how these later switches result in a higher cost and time.

Finally, we further assess how well PACT can deal with loss estimation errors, by adding a *bias* to the prediction output for the first model, namely, the full one. Fig. 10(left) shows that positive and negative biases yield similar performance decrease. Also, PACT outperforms OneSwitch even in the presence of a bias. Fig. 10(center), referring to the case $\ell^{\max}$=0.9, shows how biases on the loss variations prediction influence PACT's decisions. Consistently with Fig. 9(center), underestimating the full model's performance leads to a later switch, while overestimating it has the opposite effect. It is also worth noting the times of the *second* switch: PACT can potentially compensate for the misguided decisions it made earlier. This happens, for instance, when the bias is equal to 0.05, as K_1 and K_2 are respectively lower and larger w.r.t. the case with no bias. Fig. 10(right) underlines, similarly to Fig. 9(right), that adding a bias term leads to higher cost and time. This is a very important result, as it highlights how PACT is robust to possible errors in estimation techniques such as that demonstrated in Sec. IV.

Sequential scenario, AlexNet model. Fig. 11(left) shows the cost versus the target loss, for PACT and for each of the benchmark solutions. As in Fig. 7(left), PACT decisions are close to the optimal ones and outperform the other solutions.

Fig. 11(center) shows the loss trend as a function of the elapsed time when setting ℓ^{max}=1.1: the PACT solution takes slightly longer than the optimal one, but still less than the other two solutions. Nevertheless, in general, the time necessary to

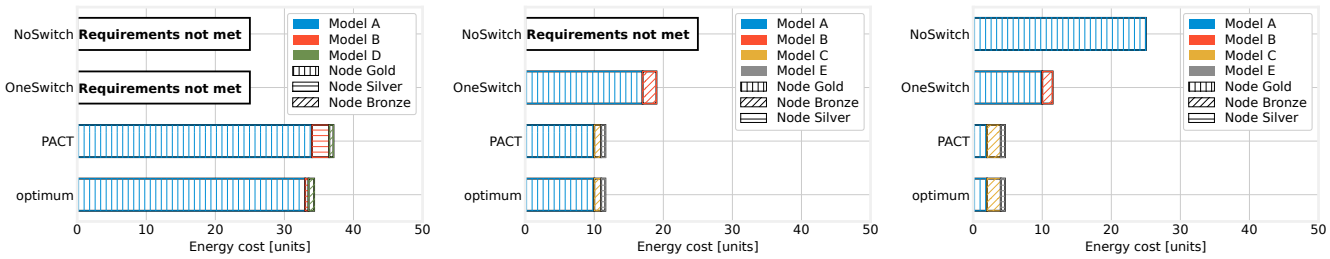

Fig. 16. PACT vs. benchmark strategies under the federated scenario with VGG: energy cost incurred by using different models when $\ell^{\max}$=1.05 (left), $\ell^{\max}$=1.2 (center), and $\ell^{\max}$=1.4 (right).

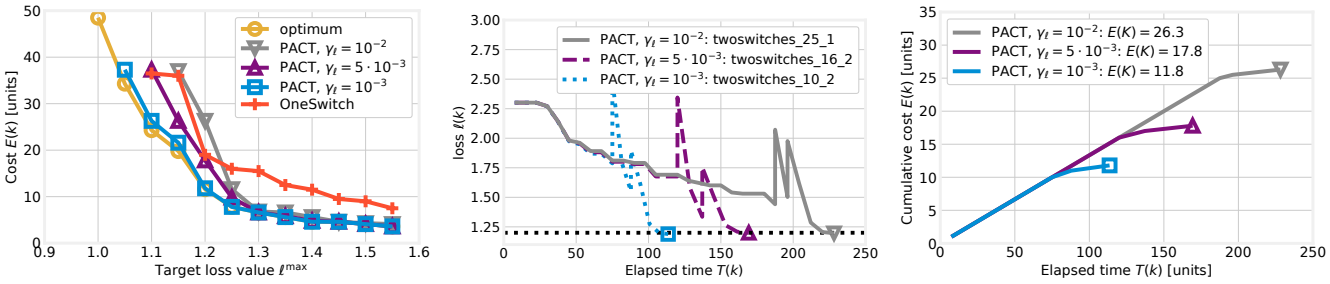

Fig. 17. Impact of γ_ℓ on PACT's performance for the federated scenario with VGG: energy cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max}$=1.2.

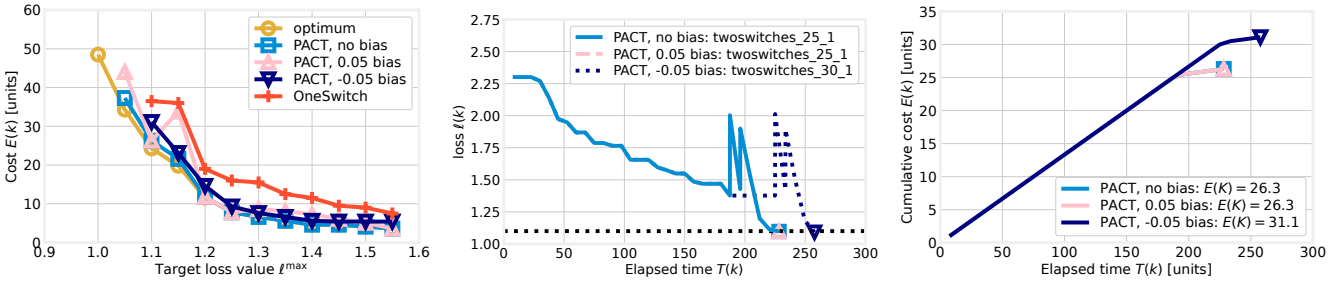

Fig. 18. Impact of quality of $\hat{\lambda}^{\text{run}}$ for the federated scenario with VGG: PACT's energy cost for different values of $\ell^{\max}$ (left); loss (center) and cost (right) evolution for $\ell^{\max}$=1.1.

achieve the desired ℓ^{max} is similar for all the solutions. On the other hand, as depicted by Fig. 11(right), the incurred cost changes among the different strategies: PACT's cost is slightly higher than the optimal one, but lower than the energy cost entailed by OneSwitch and NoSwitch.

Fig. 12, similarly to Fig. 8, shows how much the different nodes are exploited, considering three different values of $\ell^{\max}$, respectively 0.8, 1, and 1.2. When the requirements are stricter, i.e., $\ell^{\max}$ is lower, the NoSwitch and OneSwitch solutions cannot find any feasible solution and generally the energy consumption increases. Again, when the constraints are tighter, PACT requires more energy than the optimal solution.

Fig. 13(left) shows the impact of γ_ℓ: when $\ell^{\max}$ decreases, the effect of increasing γ_ℓ is noticeable and it may be not possible to reach the same $\ell^{\max}$ value as the optimum. In this case, unlike in the VGG results of Fig. 9(left), when $\ell^{\max}$ decreases, OneSwitch outperforms first the solution obtained with γ_ℓ=10^{-3} and then the one with γ_ℓ=$5 \cdot 10^{-3}$. Regarding Fig. 13(center) and Fig. 13(right), we can draw the same conclusions as for Fig. 9(center) and Fig. 9(right).

Fig. 14(left) underlines the impact of γ_T: also in this case, PACT outperforms the OneSwitch solution in the presence of a bias. Unlike Fig. 10(center), even if the effect of a bias on K_1 decision is the same, a negative bias value leads to a solution requiring less time; however, as depicted by Fig. 14(right), the energy cost always increases when applying a bias.

Federated scenario, VGG model. Fig. 15, Fig. 16, Fig. 17, and Fig. 18 depict the results obtained in the federated setting. Generally, it is possible to draw the same conclusions as for the sequential scenario. Also, Fig. 18(center) shows that for $\ell^{\max}$=1.1 applying a bias equal to 0.05 does not influence PACT's decision. Still considering the same bias value, looking at Fig. 18(left), we can notice that the trend is not monotonic. Indeed, when $\ell^{\max}$=1.15, the incurred cost is higher than for $\ell^{\max}$=1.1. This result might be counterintuitive, but the explanation is straightforward. When $\ell^{\max}$=1.15, applying a bias equal to 0.05 leads to a very inefficient K_1 choice, as it is necessary to spend a very high cost to achieve the desired loss value. Applying the same bias when $\ell^{\max}$=1.1, instead, it has a much lower effect on the decision and, thus, on the cost, which is lower than the one in the previous case.

VII. Related Work

Our work is related to four main research areas, namely: approaches to transform an ML model into a different one,

hybrid learning strategies that combine different techniques, resource-aware distributed ML, and distributed learning characterization.

Model switching and compression. It is often necessary to transform a pre- or partially-trained model into a simpler and/or smaller one, preserving (as much as possible) the learning accuracy. The two most popular techniques for model compression are knowledge distillation (KD) and pruning. Both techniques are used to achieve *model compression*, i.e., to obtain small-size models exhibiting the same learning performance as a larger one. KD [14] is a family of learning techniques predicated on the transition from a larger *teacher* model to a smaller (hence, simpler) *student* one. The student model does not learn directly from data, but mimics the behavior of the teacher model. Such learning exploits not only the decisions of the teacher model, but also the so-called *dark knowledge* embedded in its parameters. As shown in [41], additional properties such as regularization are also transferred from teacher to student models. Several works combine KD with deep reinforcement learning (RL), to learn the *value* of each *action*, as the actions themselves are taken. Studies focus on task generalization [39], in which models are transferred across nodes in charge of different tasks, and knowledge is distilled from a task to another. In the same context, [40] focuses on the heterogeneity of data at different nodes, and on mitigating its impact on learning performance. Further, [42] leverages KD to produce in a generative adversarial neural network (GANN) fashion, additional training data. [12] follows a similar approach, using KD to achieve *domain adaptation* in scenarios where no data from the source domain, but only a model trained therein, is available. A different aspect of KD is tackled by [43], [44], aiming at distilling multiple single-task policies into a single, multi-task policy.

Model pruning follows instead the more direct approach of removing some of the coefficients from a model, thus reducing its size and complexity. It is based upon the so-called *lottery ticket* hypothesis [45], assuming that very few of the parameters of a model actually impact its performance. Once such parameters can be identified, the others can be *pruned* away, resulting in a DNN with the same performance as the original one, but a much lower complexity. Several works have addressed iterative pruning where every few epochs the weights expected to least affect performance are pruned [46], [47]. A hardware-friendly scheme is proposed, e.g., in [48], with the aim to execute DNNs on low-end hardware in near real time. [20] brings privacy into the picture, highlighting how simplistic pruning may expose user data. A very effective technique is also *structured* pruning [13], which removes whole parts of a DNN (e.g., rows or columns of the parameter matrix) instead of individual ones, thus offering better efficiency, than plain pruning, at the cost of more complex decisions to make.

Pruning can be combined with KD, as envisioned in [49], where it is shown that pruned networks are easier to distill. A related problem is that pruning affects the performance of different layers to a different extent. To cope with this, [50] proposes using pruning on some DNN layers of the ResNet architecture, and compressing the others via KD. Similarly, in [51] each layer is pruned and distilled independently and in a parallel, to achieve faster distillation. Finally, recent work [52] proposes the use of RL to control pruning. In a similar spirit, [53] automatically looks for the best pruning ratio to use at every layer of a DNN, so as to simplify (hence, speed-up) learning without hurting accuracy.

Hybrid approaches. Combining different models towards a single learning task is a fairly uncommon approach. Some works explore how to alternate distributed learning schemes such as Split Learning (SL), FL, and KD. An example is [54], which splits the DNN architecture into head and tail, and replaces the former with its distilled version. [55] seeks to reduce the network delay incurred by FL by performing communication and local learning concurrently, at the price of the global model being behind local ones by several epochs. In a similar setting, [56] optimizes the computation, communication, and cooperation aspects of FL in resource-constrained scenarios. [57] leverages RL to identify the best split of a learning task (e.g., the layers of a DNN) across the available network nodes.

A related issue is *early exit*, aiming at making inference faster by skipping some of the DNN layers if a certain outcome, e.g., a certain classification decision, can be reached with high certainty. This is achieved [58] by giving the DNN a Y-shaped topology, with one branch being a lot simpler (e.g., with a smaller number of neurons) than the other. The simpler branch is ran first and, if the results are decisive enough for a certain sample, then the more complex branch can be altogether dispensed with. [59] targets highly heterogeneous scenarios where learning *one* model for all devices may indeed be suboptimal, and proposing a *personalized learning* where a different model is trained at each device; layers that are common among models are then trained in an FL fashion.

Resource-aware distributed ML. A major concern of distributed ML is *the nodes heterogeneity*, in terms of data quality, data quantity, available resources, and connectivity. In the context of FL, several works focus on *selecting* the participating nodes, accounting for their speed [60], [61], quantity [59], [59], [61], [62] and quality [62], [63] of local data, the speed and reliability of their network [55], [60] as well as trust [64]. The basic trade-off balances the need to learn more during each epoch (achieved primarily by adding more data from more nodes) with the need to shorten the epoch duration. Some studies seek to assign a trust score to each node, combining its speed and the likelihood to report inaccurate information [64]. Recent studies [65] have established a benchmark system, including datasets and helper Python classes, to easily compare FL strategies. Emulated *testbeds* for distributed ML are also emerging. A prominent example is [66], allowing the study of novel ML schemes and strategies performance over real, network-edge-class hardware. A related trend is represented by novel datasets, with tried-and-tested options like CIFAR [16] complemented by more challenging alternatives; e.g., [67] describes a dataset containing hundreds of thousands of images of numbers taken from Google Maps.

Other works, e.g., [22], [68], target a more general scenario, where DNN layers can be run, and possibly be duplicated, at different nodes. This requires balancing the opportunity to use fast learning nodes with the network delays resulting from moving data between nodes. Both works jointly consider

learning time and energy consumption: in [22] they are part of a combined objective, while in [68] the former is a constraint and the latter the sole objective. Interestingly, recent work (e.g., [69]) has aimed at creating energy-efficient DNN architectures, offering better trade-offs between energy efficiency and learning effectiveness. For instance, [69] presents variants of the MobileNet and ResNet architectures, achieving good performance with a fraction of the parameters (hence, training time) of their counterparts. Similarly, [70] proposes a family of scalable DNN architectures for sound event detection, suitable for scaling depending upon the available resources. In the same spirit, the authors of [5] seek to improve FL by dividing models into *client-* and *server*-side parts, with the former providing personalized results for local datasets, and the latter achieving high accuracy for out-of-distribution data. A different approach is adopted in [71], dropping the usage of *deep* NNs and finding instead that single-layer, *wide* NNs perform as well as their counterparts. This has an impact on how easy it is to run the learning task in parallel, as wide NNs have fewer dependencies between parameters and are thus easier to split across nodes.

Distributed learning characterization. Early work [72] studies the convergence of distributed learning, identifying the latent trade-off between involving more nodes (which means that fewer epochs are needed to converge) and exploiting fewer, faster nodes (which makes individual epochs shorter). The experiments in [28] report a power-law behavior, with the exponent depending upon the quantity of data, and the model architecture shifting the error, but not reducing the exponent itself. Other works focus on FL and derive exponential bounds on the loss [30], [73]. Some studies focus on the overparameterized regime of DNNs, investigating, e.g., how many parameters are needed for a network to be overparameterized [25], [74], Specifically, [25] finds that the number of required parameters grows quadratically with the data size (number of samples and complexity thereof). Studies focusing on KD are more rare. Examples include [75], which models the teacher-to-student translation as a price to pay on the loss, and [76] that provides a per-iteration characterization of KD.

Finally, we mention that a preliminary version of this work has been presented in our conference paper [1], where however only sequential learning was addressed and the results were obtained using a single DNN.

VIII. Conclusions

In this paper, we addressed the problem of optimizing the strategy for the distributed training of DNN models, by making joint decisions about (i) model switching (e.g., pruning), (ii) data selection, and (iii) node and resource allocation. Our objective is to minimize the energy cost, subject to learning quality and time constraints. In view of the problem NP-hardness, we proposed an algorithmic framework called PACT, following an ADP approach and leveraging both theoretical results and the outcome of previous decisions to predict the future training evolution. PACT has polynomial worst-case complexity, and the decisions it makes can be arbitrarily close to the optimum. We have further shown, through extensive numerical evaluation, that PACT outperforms state-of-the-art approaches and closely matches the optimum.

References

[1] F. Malandrino, G. Di Giacomo, A. Karamzade, M. Levorato, and C. Chiasserini, "Matching DNN compression and cooperative training with resources and data availability," in *IEEE INFOCOM*, 2023.

[2] D. Callegaro and M. Levorato, "Optimal edge computing for infrastructure-assisted uav systems," *IEEE Transactions on Vehicular Technology*, 2021.

[3] P. Tehrani, F. Restuccia, and M. Levorato, "Federated deep reinforcement learning for the distributed control of nextg wireless networks," in *IEEE DySPAN*, 2021.

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

[5] D.-J. Han, D.-Y. Kim, M. Choi, C. G. Brinton, and J. Moon, "SplitGP: Achieving Both Generalization and Personalization in Federated Learning," in *IEEE INFOCOM*, 2023.

[6] "Dnn model compression for iot domain-specific hardware accelerators," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6650–6662, 2022.

[7] H. Guo, Q. Yang, H. Wang, Y. Hua, T. Song, R. Ma, and H. Guan, "Spacedml: Enabling distributed machine learning in space information networks," *IEEE Network*, 2021.

[8] D. Callegaro, F. Restuccia, and M. Levorato, "Smartdet: Context-aware dynamic control of edge task offloading for mobile object detection," in *IEEE WoWMoM*, 2022.

[9] A. Li, L. Zhang, J. Tan, Y. Qin, J. Wang, and X.-Y. Li, "Sample-level data selection for federated learning," in *IEEE INFOCOM*, 2021.

[10] T. Tuor, S. Wang, B. J. Ko, C. Liu, and K. K. Leung, "Overcoming noisy and irrelevant data in federated learning," in *IEEE ICPR*, 2021.

[11] S. Fu, Z. Li, K. Liu, S. Din, M. Imran, and X. Yang, "Model compression for iot applications in industry 4.0 via multiscale knowledge transfer," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6013–6022, 2020.

[12] S. Tang, Y. Shi, Z. Ma, J. Li, J. Lyu, Q. Li, and J. Zhang, "Model adaptation through hypothesis transfer with gradual knowledge distillation," in *IEEE/RSJ IROS*, 2021.

[13] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, 2016.

[14] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, 2021.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

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

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[18] A. Bragagnolo and C. A. Barbano, "Simplify: A python library for optimizing pruned neural networks," *SoftwareX*, vol. 17, p. 100907, 2022.

[19] E. Yvinec, A. Dapogny, M. Cord, and K. Bailly, "RED++: Data-Free Pruning of Deep Neural Networks via Input Splitting and Output Merging," *arXiv:2110.01397*, 2021.

[20] Y. Gong, Z. Zhan, Z. Li, W. Niu, X. Ma, W. Wang, B. Ren, C. Ding, X. Lin, X. Xu *et al.*, "A privacy-preserving-oriented dnn pruning and mobile acceleration framework," in *ACM GLSVLSI*, 2020.

[21] M. Osta, M. Alameh, H. Younes, A. Ibrahim, and M. Valle, "Energy efficient implementation of machine learning algorithms on hardware platforms," in *IEEE ICECS*, 2019, pp. 21–24.

[22] C. W. Zaw, S. R. Pandey, K. Kim, and C. S. Hong, "Energy-aware resource management for federated learning in multi-access edge computing systems," *IEEE Access*, 2021.

[23] L. Ruthotto and E. Haber, "An introduction to deep generative modeling," *Wiley GAMM-Mitteilungen*, 2021.

[24] T. Abtahi, A. Kulkarni, and T. Mohsenin, "Accelerating convolutional neural network with fft on tiny cores," in *IEEE Circuits and Systems (ISCAS*, 2017, pp. 1–4.

[25] S. Oymak and M. Soltanolkotabi, "Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks," *IEEE Journal on Selected Areas in Information Theory*, 2020.

[26] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv:1312.6120*, 2013.

[27] Z. Allen-Zhu, Y. Li, and Y. Liang, "Learning and generalization in overparameterized neural networks, going beyond two layers," *Advances in neural information processing systems*, 2019.

[28] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. Patwary, Y. Yang, and Y. Zhou, "Deep learning scaling is predictable, empirically," *arXiv:1712.00409*, 2017.

[29] F. Altché and A. de La Fortelle, "An LSTM network for highway trajectory prediction," in *IEEE Transportation Systems (ITSC*, 2017, pp. 353–359.

[30] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on non-iid data," in *ICLR*, 2020.

[31] D. G. Cattrysse and L. N. Van Wassenhove, "A survey of algorithms for the generalized assignment problem," *European Journal of Operational Research*, 1992.

[32] M. L. Fredman, "New bounds on the complexity of the shortest path problem," *SIAM Journal on Computing*, 1976.

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

[34] "NVIDIA A100 datasheet," https://bit.ly/3O6jWxj, accessed: 2021-07-30.

[35] "NVIDIA RTX A4000 datasheet," https://bit.ly/3ceeUS8, accessed: 2021-07-30.

[36] "Broadcom VideoCore VI technical details," https://bit.ly/3z5bytK, accessed: 2021-07-30.

[37] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[38] F. Rodrigues and F. C. Pereira, "Beyond expectation: Deep joint mean and quantile regression for spatiotemporal problems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 12, pp. 5377–5389, 2020.

[39] Z. Gao, K. Xu, B. Ding, H. Wang, Y. Li, and H. Jia, "Knowru: Knowledge reusing via knowledge distillation in multi-agent reinforcement learning," *arXiv:2103.14891*, 2021.

[40] T. Zhang, X. Wang, B. Liang, and B. Yuan, "Catastrophic interference in reinforcement learning: A solution based on context division and knowledge distillation," *arXiv:2109.00525*, 2021.

[41] L. Saglietti and L. Zdeborová, "Solvable model for inheriting the regularization through knowledge distillation," *arXiv:2012.00194*, 2020.

[42] Y. Huang and Y. Yu, "Distilling deep neural networks with reinforcement learning," in *IEEE ICIA*, 2018.

[43] H. Yin and S. J. Pan, "Knowledge transfer for deep reinforcement learning with hierarchical experience replay," in *Thirty-First AAAI conference on artificial intelligence*, 2017.

[44] Z.-W. Hong, P. Nagarajan, and G. Maeda, "Collaborative inter-agent knowledge distillation for reinforcement learning," 2019.

[45] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *ICLR*, 2018.

[46] C. M. J. Tan and M. Motani, "Dropnet: Reducing neural network complexity via iterative pruning," in *ICML*, 2020.

[47] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *ECCV*, 2018.

[48] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning," in *ACM ASPLOS*, 2020.

[49] J. Park and A. No, "Prune your model before distill it," *arXiv:2109.14960*, 2021.

[50] N. Aghli and E. Ribeiro, "Combining weight pruning and knowledge distillation for cnn compression," in *IEEE/CVF CVPR Workshops CVPRW*, 2021.

[51] T. Menzies, S.-H. Lim, H. Guan, X. Shen, and L. Ning, "Generalization of teacher-student network and cnn pruning," North Carolina State University. Dept. of Computer Science, Tech. Rep., 2018.

[52] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *ECCV*, 2018.

[53] Y. Sakai, Y. Eto, and Y. Teranishi, "Structured pruning for deep neural networks with adaptive pruning rate derivation based on connection sensitivity and loss function," *Journal of Advances in Information Technology*, 2022.

[54] Y. Matsubara, D. Callegaro, S. Baidya, M. Levorato, and S. Singh, "Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems," *IEEE Access*, 2020.

[55] Y. Zhou, Q. Ye, and J. C. Lv, "Communication-Efficient Federated Learning with Compensated Overlap-FedAvg," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[56] A. Chopra, S. K. Sahu, A. Singh, A. Java, P. Vepakomma, V. Sharma, and R. Raskar, "Adasplit: Adaptive trade-offs for resource-constrained distributed deep learning," *arXiv:2112.01637*, 2021.

[57] T. Sen and H. Shen, "A data and model parallelism-based distributed deep learning system in a network of edge devices," in *ICDCS*, 2022.

[58] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *arXiv:2103.04505*, 2021.

[59] O. Marfoq, G. Neglia, R. Vidal, and L. Kameni, "Personalized federated learning through local memorization," in *ICML*, 2022.

[60] 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.

[61] A. M. Abdelmoniem, A. N. Sahu, M. Canini, and S. A. Fahmy, "Resource-Efficient Federated Learning," *arXiv:2111.01108*, 2021.

[62] F. Malandrino and C. F. Chiasserini, "Federated learning at the network edge: When not all nodes are created equal," *IEEE Communications Magazine*, 2021.

[63] H. Wu and P. Wang, "Fast-convergent federated learning with adaptive weighting," *IEEE Transactions on Cognitive Communications and Networking*, 2021.

[64] A. Imteaj and M. H. Amini, "Fedar: Activity and resource-aware federated learning model for distributed mobile robots," in *IEEE ICMLA*, 2020.

[65] F. Lai, Y. Dai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "FedScale: Benchmarking model and system performance of federated learning," in *ACM SIGOPS ResilientFL Workshop*, 2021.

[66] L. Yang, F. Wen, J. Cao, and Z. Wang, "Edgetb: A hybrid testbed for distributed machine learning at the edge with high fidelity," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2540–2553, 2022.

[67] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[68] F. Malandrino, C. F. Chiasserini, and G. Di Giacomo, "Efficient distributed dnns in the mobile-edge-cloud continuum," *IEEE/ACM Transactions on Networking*, 2023.

[69] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, 2019.

[70] F. Paissan, A. Ancilotto, A. Brutti, and E. Farella, "Scalable neural architectures for end-to-end environmental sound classification," in *IEEE ICASSP*, 2022.

[71] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 215–223.

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

[73] N. Zeulin, O. Galinina, N. Himayat, S. Andreev, and R. W. Heath Jr, "Dynamic Network-Assisted D2D-Aided Coded Distributed Learning," *arXiv:2111.14789*, 2021.

[74] Z. Chen, Y. Cao, D. Zou, and Q. Gu, "How much over-parameterization is sufficient to learn deep relu networks?" in *ICLR*, 2021.

[75] M. Phuong and C. Lampert, "Towards understanding knowledge distillation," in *PMLR International Conference on Machine Learning*, 2019.

[76] A. Rahbar, A. Panahi, C. Bhattacharyya, D. Dubhashi, and M. H. Chehreghani, "On the unreasonable effectiveness of knowledge distillation: Analysis in the kernel regime," *arXiv:2003.13438*, 2020.