Matching DNN Compression and Cooperative Training with Resources and Data Availability

F. Malandrino[1,2], G. Di Giacomo[3], A. Karamzade[4], M. Levorato[4], C. F. Chiasserini[3,1,2]
1: CNR-IEIIT, Italy – 2: CNIT, Italy – 3: Politecnico di Torino, Italy – 4: UC Irvine, USA

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10229076

***Abstract*—To make machine learning (ML) sustainable and apt to run on the diverse devices where relevant data is, it is essential to compress ML models as needed, while still meeting the required learning quality and time performance. However, *how much* and *when* an ML model should be compressed, and *where* its training should be executed, are hard decisions to make, as they depend on the model itself, the resources of the available nodes, and the data such nodes own. Existing studies focus on each of those aspects individually, however, they do not account for how such decisions can be made jointly and adapted to one another. In this work, we model the network system focusing on the training of DNNs, formalize the above multi-dimensional problem, and, given its NP-hardness, formulate an approximate dynamic programming problem that we solve through the PACT algorithmic framework. Importantly, PACT leverages a time-expanded graph representing the learning process, and a data-driven and theoretical approach for the prediction of the loss evolution to be expected as a consequence of training decisions. We prove that PACT's solutions can get as close to the optimum as desired, at the cost of an increased time complexity, and that, in any case, such complexity is polynomial. Numerical results also show that, even under the most disadvantageous settings, PACT outperforms state-of-the-art alternatives and closely matches the optimal energy cost.**

I. Introduction

Modern-day machine-learning (ML) models are hard to train, as they often require considerable quantities of data as well as computational, network and energy resources [1], [2]. In addition, in several application scenarios, data and resources may be located across different network nodes, whose availability and connectivity may significantly differ, and even vary in space and time [3]. Examples include smart factory ML-based applications, where models are partially trained in the cloud and then specialized by edge nodes [4], or even more extreme settings where image classification models are first trained by ground stations and then refined by a spacecraft specifically for its operating environment [5].

In all the above cases, a critical *technical challenge* is the *mutual adaptation of the training process and the system settings, resources and data* offered by the interconnected network nodes. We contend that existing works only address some specific aspects but none of them tackles the joint optimization of ML model compression and the selection of nodes and related data. The framework we propose achieves such goals by leveraging *multiple* ML models throughout different stages of a single learning task – *switching* among them as needed (e.g., via model pruning [6], or knowledge distillation (KD) [7]) – and choosing, for each stage, the most appropriate datasets and resources. To make an example, the training of a complex model can start over a small set of powerful nodes; then, we can switch to a simpler, compressed model and perform further training epochs involving additional nodes that contribute fewer resources but more valuable data [8] toward their specific task and domain. Model and nodes switching, however, comes with its own costs in terms of time, resources and, often, learning performance, hence, switching decisions must be made only when clear gains can be obtained.

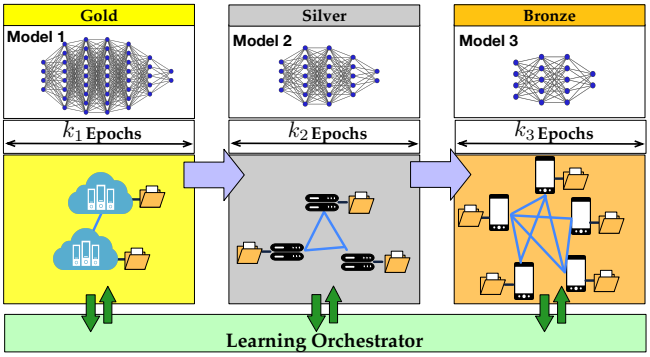

Fig. 1. Cooperative training process proposed and optimized in this paper. Subsets of nodes sequentially train compressed versions of an original DNN model. In the picture, we categorize nodes based on their computing capabilities and data availability, and, in the example, the training sequence is based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

In this work, we formalize and optimize such complex cooperative strategies for the training of deep neural networks (DNNs), where the – heterogeneous – nodes that participate in the training do not share their data. The training process is illustrated in Fig. 1. In this challenging scenario, we make the following contributions:

(1) *Model and problem definition:* We develop a model for the networked system supporting the training of DNN models capturing its most relevant aspects, and we use it to make *dynamic, joint* decisions about: (i) the ML *model* to be used at each epoch (e.g., the full DNN model or a compressed version thereof); (ii) *when* to perform a model switch (e.g., at which epoch the framework transitions to a compressed version of the DNN); and (iii) the *nodes* to leverage at each stage, accounting for their resources and specific local datasets. The overarching goal is to reach a target learning quality by a desired deadline, while minimizing the energy consumption (hence, the cost) of the overall process. We remark that controlling *all* the above aspects allows for more flexibility than existing works that just study how to adapt one aspect (e.g., choosing the model) to another (e.g., a given and immutable set of resources).

(2) *Algorithmic framework:* The decision process is further complicated by two main issues, namely, (i) the scale of the problem itself, and (ii) the fact that the effect of model switching decisions cannot be, in general, known with certainty. We tackle the first issue by adopting an approximate dynamic programming (ADP) approach, whereby only the most promising courses of action are evaluated, and we envision an algorithmic solution, named Performance-Aware Compression and Training (PACT), for an efficient selection thereof. Concerning the second, we integrate into our decision-making approach the available results on performance characterization of DNN model training. In particular, we leverage both theoretical bounds and data-driven predictions based on low-complexity NN architectures of the loss evolution to be expected as a consequence of a sequence of training decisions. Importantly, we prove that PACT can get as close as desired to the optimum of the formulated problem (which is shown to be NP-hard), at the cost of an increased complexity, which is anyway polynomial at worst.
(3) *Performance evaluation:* We show how PACT identifies the training strategy that best matches the available resources and data, resulting in minimum energy consumption given the target loss value and training process time. Also, PACT demonstrates to be highly robust to approximate estimations of the effects of model switching on the loss values.

In the rest of the paper, 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 used 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 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 [6]. 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 VGG11 DNN model [9] 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 [10];
- then, a fraction F of the model's parameters is pruned
- finally, training resumes adding 2 more learning nodes, which have 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.

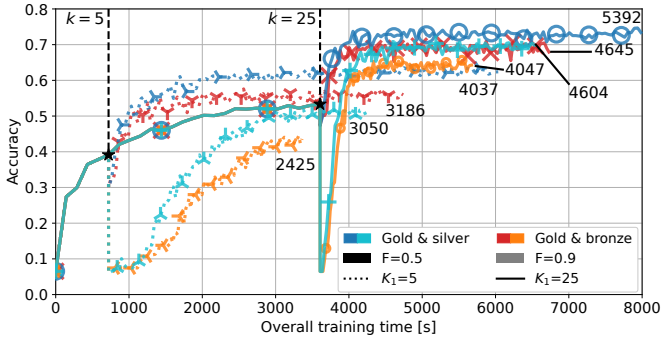

Fig. 2. Accuracy vs. training time for different values of pruning epoch K_1 (denoted by different line styles) and percentage F (denoted by different color shades). Note how upon pruning a sudden drop in accuracy occurs. Cold and warm colors denote the set of nodes used for FL. Numbers in the plot indicate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

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[1], 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 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 build the representation of the system we tackle, and formulate the problem of optimally matching DNN compression and training with resources/data availability.

[1]Only some values of F are possible, as we apply structured pruning (see Sec. VII for further details).

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 loss function at k, 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 ∞.

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 we enact at epoch k. However, now 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 [11], [12]. Impossible transitions 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.

B. Problem definition

Given the impelling need to make ML sustainable [13], [14], our goal is to minimize the overall learning energy consumption, while ensuring that the loss drops below a target value $\ell^{\max}$ within time $T^{\max}$. 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 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 [15]. 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 present the methodology used at the learning orchestrator in order to compute them.

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 [16]–[18] and experimental [19] works. Then, we can write the

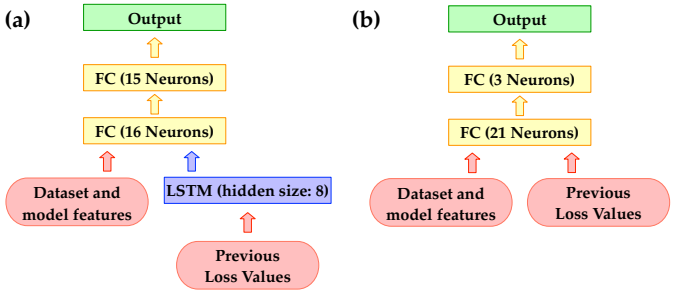

Fig. 3. Architecture of the $\hat{\lambda}^{\text{run}}$ (a) and $\hat{\lambda}^{\text{change}}$ (b) estimators.

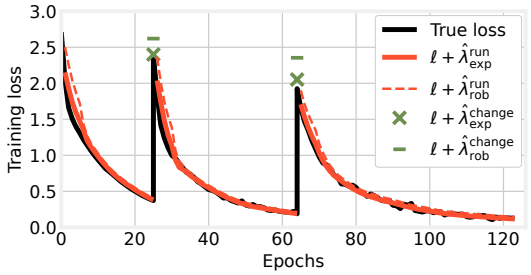

Fig. 4. Example of true loss vs expected-value and robust estimators.

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:

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

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

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) 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 training loss variation as well as determine the prediction uncertainty. Specifically, for $\lambda^{\text{run}}(k,m,n)$, we take as a starting point the Long Short-Term Memory (LSTM) model in [20] and develop a similar, yet simpler, branched architecture, as depicted in Fig. 3(a). 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 NN 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 of changing models, 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.

As depicted in Fig. 4 the above estimators produce very accurate predictions (red lines and green markers for λ^{run} and λ^{change}, resp.) of the true loss (black line). The figure refers to the training loss of the VGG11 DNN model initially trained with 45,000 samples of the CIFAR-10 dataset with 10 classes. After 25 epochs, the model is pruned with $F_1 = 0.75$ and handed over to a set of nodes owning 5,000 samples belonging to 13 classes, taken from the CIFAR-10 and CIFAR-100 datasets. Likewise, after 40 more epochs, the model is further pruned with $F_2 = 0.5$, and passed to a third set of nodes owing 1,500 samples belonging to 15 classes, taken from the same combination of datasets.

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 $\lambda^{\text{run}}(k,m,n)$ [21, Theorem 1] and for $\lambda^{\text{change}}(k,m,m')$ [11, 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 $\gamma_\ell = 0.1$, a vertex with $\ell = 0.1$ or 0.2 can exist, but not with $\ell = 0.15$);
- 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 $\gamma_T = 0.1$ and $\gamma_\ell = 0.1$. 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: $\quad\mathcal{V} \leftarrow \{\Omega\}$ ▷ *set of vertices*
3: $\quad$**for all** $m \in \mathcal{M}, n \in \mathcal{N}$ **do**
4: $\quad\quad$**for all** $k \in \left[1, 2, \ldots, \left\lceil \frac{T^{\max}}{\gamma_T} \right\rceil\right]$ **do**
5: $\quad\quad\quad$**for all** $\ell \in [0, \gamma_\ell, 2\gamma_\ell, \ldots, \ell(0)]$ **do**
6: $\quad\quad\quad\quad$**for all** $T \in [0, \gamma_T, 2\gamma_T, \ldots, T^{\max}]$ **do**
7: $\quad\quad\quad\quad\quad v \leftarrow (k, m, n, \ell, T)$
8: $\quad\quad\quad\quad\quad \mathcal{V} \leftarrow \mathcal{V} \cup \{v\}$
9: $\quad\mathcal{E} \leftarrow \emptyset$ ▷ *set of edges*
10: $\quad$**for all** $v = (k, m, n, \ell, T) \in \mathcal{V}$ **do**
11: $\quad\quad$**for all** $\mathbf{a} = (m', n') \in \mathbf{A}$ **do**
12: $\quad\quad\quad T' \leftarrow T + \tau^{\text{change}}(m, n, m', n') + \tau^{\text{run}}(m, n)$
13: $\quad\quad\quad E \leftarrow \epsilon^{\text{change}}(m, n, m', n') + \epsilon^{\text{run}}(m', n')$
14: $\quad\quad\quad$**if** $T' > T^{\max} \vee E = \infty$ **then**
15: $\quad\quad\quad\quad$**continue** ▷ *infeasible, skip this action*
16: $\quad\quad\quad \ell' \leftarrow \ell + \hat{\lambda}^{\text{change}}_{\text{rob}}(k, m, m') + \hat{\lambda}^{\text{run}}_{\text{rob}}(k, m, n')$
17: $\quad\quad\quad \ell' \leftarrow \gamma_\ell \left\lceil \frac{\ell'}{\gamma_\ell} \right\rceil$
18: $\quad\quad\quad T' \leftarrow \gamma_T \left\lceil \frac{T'}{\gamma_T} \right\rceil$
19: $\quad\quad\quad v' \leftarrow (k+1, m', n', \ell', T')$
20: $\quad\quad\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(v, v', \mathsf{weight} = E)\}$
21: $\quad\quad$**if** $\ell \leq \ell^{\max} \wedge T \leq T^{\max}$ **then**
22: $\quad\quad\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{v, \Omega\}$ ▷ *feasible state*
23: $\quad$**return** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

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,

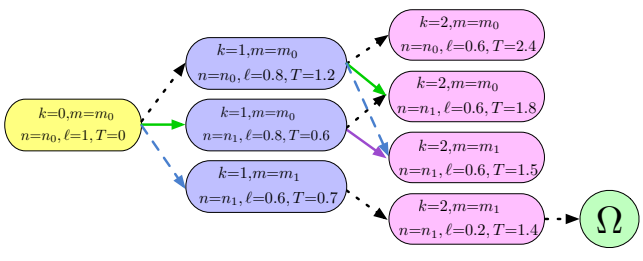

Fig. 5. Example of the expanded graph generated by PACT, with resolution values $\gamma_T = 0.1$ and $\gamma_\ell = 0.1$, learning target $\ell^{\max} = 0.25$, and time limit $T^{\max} = 1.5$. Edge colors correspond to switches made across subsequent epochs: node only (solid green), model only (solid 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** $\mathcal{P}$

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 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) [22], which is known to be NP-hard. Furthermore, we prove that:

Property 2: PACT's time complexity is polynomial.

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: $\texttt{Wr} \leftarrow \min_{\overline{v} \in \overline{\mathcal{V}}} \text{weight}(\textbf{shortestPath}(p[1], \Omega, \text{via } \overline{v})$
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** $\mathbf{a} = (p[1][1], p[1][2])$

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 [23]) 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 [23].

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

Property 3: 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 3 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.

VI. PERFORMANCE RESULTS

We first describe in Sec. VI-A how we implement the loss prediction. Then we compare the performance of PACT against the optimum and state-of-the-art approaches in Sec. VI-C, under the scenario and settings described in Sec. VI-B.

A. Loss prediction implementation

To collect the training loss data necessary for the training of the estimators for λ^{change} and λ^{run}, we consider a scenario with three sets of nodes: the gold set has 45,000 samples of the CIFAR-10 belonging to 10 classes; the silver one has 5,000 samples out of the CIFAR-10 and CIFAR-100 datasets, belonging to 13 classes; and the bronze set has 1,500 samples out of the CIFAR-10 and CIFAR-100 datasets, belonging to 15 classes. Experiments always start with the gold set of nodes training a full model. Then both the cases of one and two pruning occurrences (i.e., two and three models) are considered, with pruning being performed as described in Sec. II. In the former case, after K_1 epochs, the model is pruned with pruning ratio F_1 and hand it over to the silver or the bronze set, which continues the training. In the second case, after the second set of nodes trains the pruned model for K_2 more epochs, and then prunes the model again with ratio F_2, it sends it to the last set of nodes, which completes the training. Experiments have been run for all combinations of pruning/training and of the involved parameters; specifically, we have considered: $K_1, K_2 \in [5, 15, 25, 40, 60, 100]$, $F_1 \in [0.5, 0.75]$ and $F_2 \in [0.25, 0.5]$. Notice that these are the combinations we leverage for our training, and do not limit in any way the decisions that PACT or its benchmarks can make. For the training of the NNs used for prediction, we used the Adam optimizer with learning rate of 10^{-3}, and set the batch size to 16. The loss is given by the Mean Square Error (MSE), with titled loss to compute the quantiles [24].

To assess the quality of prediction, we evaluate: the Mean Absolute Error (MAE), the Mean Interval Length (MIL) (i.e., the average width of the prediction interval), and the Interval Coverage Percentage (ICP) (i.e., the fraction of true values falling within the relative prediction interval), with the latter two indicating the quality of the quantiles prediction. The excellent results we get are presented in Tab. I, which reports the mean and the standard deviation of the three metrics, computed over all possible combinations of pruning and training configurations, and executing 10 runs for each of such cases. MIL and ICP are calculated for the 90% prediction interval, as the considered quantiles are 0.05 and 0.95.

B. Reference scenario

To assess PACT's performance, we focus on a smart factory-based application using the VGG11 DNN. Three models are considered (called *L*, *M*, and *S*), corresponding to (resp.) a full DNN, a DNN pruned with $F = 0.5$, and a DNN further pruned with $F = 0.75$. Again, we consider that *gold*, *silver*, and *bronze* sets of nodes are available, located (resp.) in the cloud, in the far edge of the network infrastructure, and in the near edge covering the smart-factory premises. To match the model size with the nodes' capability, each model best runs on one of the sets, hence, switching between models also implies changing the set of nodes to use. To reflect the real-world capabilities of (resp.) NVIDIA Ampere A100 [25] (gold nodes), NVIDIA RTX A4000 [26] (silver nodes), and Raspberry Pi's Videocore 6 [27] (bronze nodes) GPUs, (i) the duration and energy cost required for M to be trained by the silver set for one epoch are one fifth and a half (resp.) smaller than those experienced when the L model is trained by the gold set, and (ii) such values reduce to a half and to one fifth (resp.) when S is trained by the bronze nodes. Further, for simplicity, we set a very long time limit of $T^{\max} = 1,000$ time units.

TABLE I
LOSS PREDICTION: MEAN AND STANDARD DEVIATION

Model	MAE	MIL	ICP
$\hat{\lambda}^{\text{change}}$	0.13 ± 0.01	0.52 ± 0.05	0.87 ± 0.05
$\hat{\lambda}^{\text{run}}$	0.0173 ± 0.0004	0.078 ± 0.003	0.90 ± 0.01

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) *StaticLearn*: model switching is made so as to obtain similar loss decrease under all three models; (iii) *OneSwitch*: only two models are used. For both the *StaticLearn* and *OneSwitch* solution, we consider the best decisions they yield for each value of $\ell^{\max}$. Specifically, for *StaticLearn*, we consider the lowest energy cost, feasible strategy obtaining similar (within 5% margin) loss improvement from the three models. 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 [7], [28], [29] envision pruning once, hence, their performance would be represented by *OneSwitch*.

C. PACT performance

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. 6(left) shows the cost associated with each strategy, for different loss targets $\ell^{\max}$. We can observe that, 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 outperforms the alternative solutions, and closely matches the optimum. Also, only switching models once has the worst performance, a sign that switching across *multiple* models and nodes is indeed beneficial when learning constraints are tight.

Fig. 6(center) depicts the time evolution of the loss $\ell(k)$ for $\ell^{\max} = 0.15$. We can notice a power-law behavior – well captured by our predictors (see Fig. 4) and consistent with existing literature [19], [30], combined with the peaks due to the loss variation incurred when switching models. Remarkably, PACT makes virtually the same decisions as the optimum, i.e., performs the model switching at (almost) the same times. OneSwitch can only switch once, hence, does so later. As for StaticLearn, in order to achieve similar loss gains under all models, it has to switch from L to M earlier than it should, and from M to S later, achieving the learning target much later than the alternatives. Interestingly, PACT achieves the learning target earlier than the optimum, which would seem counterintuitive until we recall that *cost* is the optimization

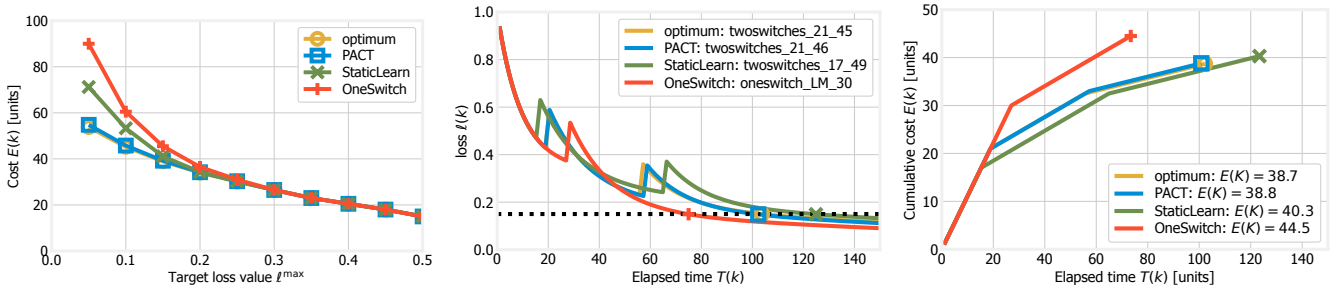

Fig. 6. PACT and benchmark strategies: cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max} = 0.15$.

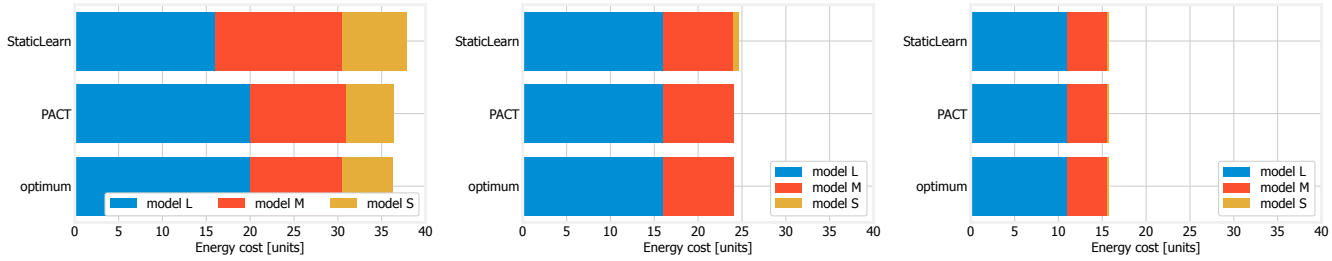

Fig. 7. PACT vs. benchmark strategies: energy cost incurred by using different models when $\ell^{\max} = 0.15$ (left), $\ell^{\max} = 0.3$ (center), $\ell^{\max} = 0.45$ (right).

objective (5), while time is a mere constraint. Accordingly, Fig. 6(right) highlights how the optimum indeed takes slightly longer than PACT to reach the objective but does so at a (marginally) lower cost (see the position of the last marker on the y-axis). This also underlines the importance of making joint decisions about learning and networking aspects, e.g., in this case, to consider both the performance of the models and the cost of the nodes they run on.

Fig. 7 sheds further light on how different strategies use the network infrastructure. Plots therein show how much energy is spent running each of the three models under the optimum, PACT, and StaticLearn strategies; different plots correspond to different values of $\ell^{\max}$. Consistently with Fig. 6(left), when $\ell^{\max}$ is high or moderate, all strategies make very similar decisions. When $\ell^{\max}$ is low, as in Fig. 7(left), the difference between PACT and StaticLearn emerges more clearly; interestingly, the former spends more energy using model L and less using model S. Notice that model L has the *highest* cost, hence, one would expect it to be wise to use that for as short as possible. Instead, both PACT and the optimum correctly account for the fact that the quicker learning progress occurring under that model compensates the higher cost it incurs. Even more interestingly, when $\ell^{\max}$ is low, PACT and the optimum do not use model S, i.e., they only switch *once*. This is consistent with the fact that, as per Fig. 6(left), PACT and OneSwitch have the same performance, and highlights the flexibility of PACT in deciding not only *when* to switch models, but also on *whether* to do so.

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. 8(left) shows that, while a larger value of γ_ℓ does indeed decrease PACT's performance, such an effect is limited: even increasing γ_ℓ by an order of magnitude does not impact PACT's ability to outperform StaticLearn, especially in the most challenging cases when $\ell^{\max}$ is small.

Fig. 8(center), referring to the case $\ell^{\max} = 0.15$, 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. 8(right) highlights how these later switches result in a higher cost – though, similarly to Fig. 6(right), *not* necessarily in a longer learning time.

Finally, we further assess how well PACT can deal with loss estimation errors, by adding a *bias* to the prediction output for model L. Fig. 9(left) shows that positive and negative biases yield similar performance decrease. Also, except for the simple cases when $\ell^{\max}$ is very large, PACT outperforms StaticLearn even in the presence of a bias. Fig. 9(center), referring to the case $\ell^{\max} = 0.15$, shows how biases on the loss variations prediction influence PACT's decisions. Consistently with Fig. 8(center), underestimating L'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, from M to S: PACT can leverage its bias-free knowledge of the performance of M and S, compensating for the misguided decisions it made earlier. Fig. 9(right) underlines, similarly to Fig. 8(right), that switching earlier results in a longer training time, though this does not necessarily result in a higher energy cost.

VII. Related Work

Model switching and compression. The two most popular techniques for model compression are KD and pruning. In KD [7], a small-size (student) model does not learn directly from data, but mimics the behavior of the large (teacher)

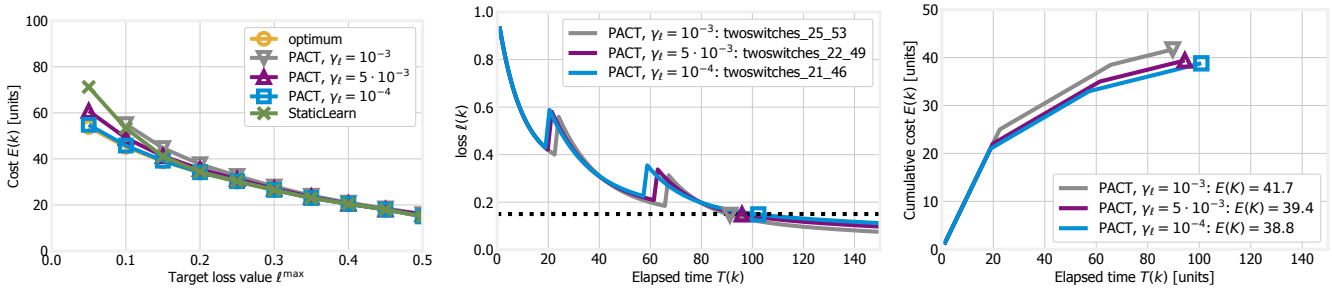

Fig. 8. Impact of γ_ℓ on PACT's performance: energy cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max} = 0.15$.

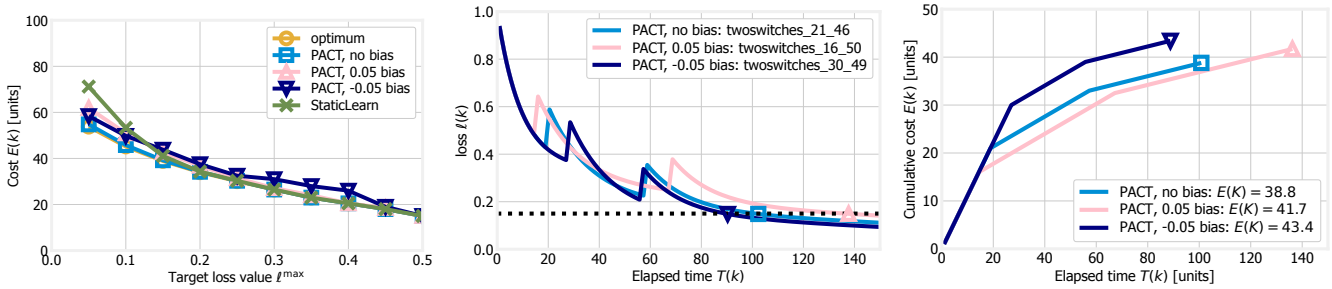

Fig. 9. Impact of quality of $\hat{\lambda}^{\text{run}}$: PACT's energy cost for different values of $\ell^{\max}$ (left); loss (center) and cost (right) evolution for $\ell^{\max} = 0.15$.

model. Studies focus on task generalization [28], and data heterogeneity [29]. As for pruning, a very effective technique is *structured* pruning [6], which removes whole parts of a DN (e.g., rows or columns of the parameter matrix). Finally, recent work [31] proposes the use of RL to control pruning.

Hybrid approaches. Some works explore how to alternate distributed learning schemes such as Split Learning (SL), FL, and KD. An example is [32], which splits the DNN architecture into head and tail, and replaces the former with its distilled version. [33] seeks to reduce the network delay incurred by FL by performing communication and local learning concurrently. In a similar setting, [34] optimizes the computation, communication, and cooperation aspects of FL in resource-constrained scenarios.

Resource-aware distributed ML. In the context of FL, several works focus on *selecting* the participating nodes, accounting for their speed [35], [36], quantity [36], [37], [37], [38] and quality [38], [39] of local data, the speed and reliability of their network [33], [35] as well as trust [40]. The basic trade-off balances the need to learn more during each epoch with the need to shorten the duration of epochs. Other works [14], [41] 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. Interestingly, recent work (e.g., [42]) has aimed at creating energy-efficient DNN architectures, offering better trade-offs between energy efficiency and learning effectiveness.

Distributed learning characterization. Early work [43] studies the convergence of distributed learning, identifying the latent trade-off between involving more nodes and exploiting fewer, faster nodes. The experiments in [19] report a power-law behavior, with the exponent depending on 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 [21], [30]. Studies focusing on KD are more rare. Examples include [44], which models the teacher-to-student translation as a price to pay on the loss, and [45] that provides a per-iteration characterization of KD.

VIII. Conclusions

We addressed the problem of matching the training and compression of DNN models, with the aim to minimize energy consumption while meeting learning performance and system constraints. To do so, we used approximate dynamic programming and developed the PACT algorithmic framework to overcome the problem's NP-hardness. PACT uses a time-expanded graph to model the system and leverages both a data-driven and a theoretical approach for predicting the loss behavior as training decisions are made. Results show that PACT matches the minimum energy consumption very closely (with worst-case polynomial complexity), while meeting the learning quality and time requirements.

Acknowledgement

Parts of this work have been performed in the framework of the Horizon Europe project CENTRIC (Grant No. 101096379). It was also supported by the European Union's NextGenerationEU instrument, under the Italian National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.3, enlarged partnership "Telecommunications of the Future" (PE0000001), program "RESTART". Marco Levorato's and Armin Karamnzade's work was partially supported by NSF under grants CNS 2134567 and CNS 2003237.

REFERENCES

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

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

[3] D. Callegaro, F. Restuccia, and M. Levorato, "Smartdet: Context-aware dynamic control of edge task offloading for mobile object detection," *arXiv preprint arXiv:2201.04235*, 2022.

[4] E. Russo, M. Palesi, S. Monteleone, D. Patti, A. Mineo, G. Ascia, and V. Catania, "Dnn model compression for iot domain-specific hardware accelerators," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6650–6662, 2022.

[5] 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.

[6] 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.

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

[8] 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.

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

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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 preprint arXiv:1712.00409*, 2017.

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

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

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

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

[24] 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.

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

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

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

[28] 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 preprint arXiv:2103.14891*, 2021.

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

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

[31] 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.

[32] 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.

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

[34] 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 preprint arXiv:2112.01637*, 2021.

[35] 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.

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

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

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

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

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

[41] 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.

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

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

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

[45] 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 preprint arXiv:2003.13438*, 2020.