Dependable Distributed Training of Compressed Machine Learning Models

Dependable Distributed Training of Compressed Machine Learning Models

Francesco Malandrino*†, Giuseppe Di Giacomo‡, Marco Levorato§, Carla Fabiana Chiasserini‡*†
*CNR-IEIIT, Italy – †CNIT, Italy – ‡Politecnico di Torino, Italy – §UC Irvine, USA

***Abstract*—The existing work on the distributed training of machine learning (ML) models has consistently overlooked the *distribution* of the achieved learning quality, focusing instead on its average value. This leads to a poor *dependability* of the resulting ML models, whose performance may be much worse than expected. We fill this gap by proposing DepL, a framework for *dependable learning orchestration*, able to make high-quality, efficient decisions on (i) the data to leverage for learning, (ii) the models to use and when to switch among them, and (iii) the clusters of nodes, and the resources thereof, to exploit. For concreteness, we consider as possible available models a full DNN and its compressed versions. Unlike previous studies, DepL guarantees that a target learning quality is reached *with a target probability*, while keeping the training cost at a minimum. We prove that DepL has constant competitive ratio and polynomial complexity, and show that it outperforms the state-of-the-art by over 27% and closely matches the optimum.**

I. Introduction

Machine learning (ML) models, and deep neural networks (DNNs) in particular, are becoming more capable, but also harder and more costly to train. This issue has been tackled through two complementary approaches: distributed training and model compression. The former exploits the resources (e.g., computational) and data available at different nodes, thus better distributing the training burden. The latter includes a wealth of different techniques (e.g., model pruning and knowledge distillation) allowing for the transfer of knowledge from a model to a (usually, simpler) one.

Distributed training and model compression have been variously combined together in the literature, allowing the used resources to adapt to the chosen model [1], selecting the best model given the available resources [2], [3], and even performing mutual adaptation between the two [4]. However, an aspect that has been overlooked so far, which we aim to investigate in this work, is the *dependability* of DNN training. Indeed, ML is increasingly used for mission-critical (and even safety-related) applications, hence traditional approaches focusing on the expected learning quality may be inadequate to present needs.

In this work, we aim at providing the right network support to make DNN training *dependable*, i.e., to guarantee that a target learning quality (e.g., loss or accuracy) is reached by a target time and *with a target probability* – at the minimum cost. To this end, we make the key observation that learning dependability is affected by two main factors: (i) the used *models* (i.e., full or compressed versions), and (ii) the quality of *training* (including the data generated by data sources as well as the nodes and the resources thereof that are used).

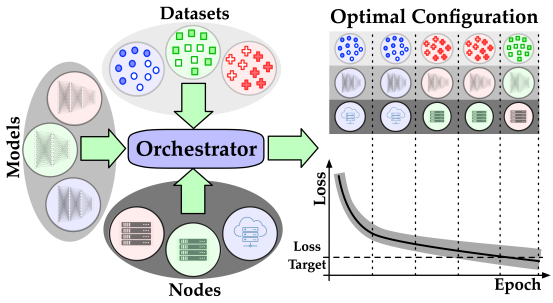

Fig. 1. DepL enables the optimization of dependable, distributed training of neural networks over heterogeneous nodes, datasets, and model architectures. The optimal configuration is designed to control the uncertainty in the loss progression along training epochs.

Our work is the first to account for all these contributions in a holistic manner, and to leverage them all to make high-quality, efficient training decisions. Importantly, although we focus on model compression as the source of alternative models, the problem we pose and the solution we envision extend to arbitrary models.

More specifically, we consider a distributed learning scenario where nodes in different parts of the network (e.g., far edge, edge, and cloud) cooperatively train a DNN model aided by a learning orchestrator. In this setting, we develop and evaluate a novel DNN training scheme called DepL (for *dep*endable *l*earning), allowing the learning orchestrator to make joint decisions about: (a) the data sources (hence, the datasets) to exploit for learning; (b) the models to train for the ML task at hand, and when to switch from one to another; (c) the node clusters to use at different stages of training. Unlike previous work, DepL accounts for the mutual interaction between such decisions, and the effect that each of them has on the expected learning quality *as well as its distribution*. Thanks to this ability, DepL can meet the challenges of present- and next-generation ML applications, including safety-critical ones. Also, DepL makes near-optimal decisions, minimizing the cost incurred by the learning process, while keeping a remarkably low (polynomial) computational complexity. Finally, DepL can be combined with emerging approaches based upon *conformal prediction sets* [5], [6], seeking to add reliability *a posteriori*, i.e., starting from an already-trained model. Indeed, models trained in a dependable manner through DepL also have smaller conformal sets.

The main contributions of our work are thus as follows:

• We propose a comprehensive system model, capturing all relevant aspects of distributed ML training, including learning

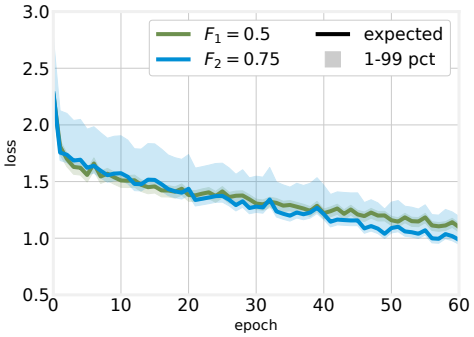

Fig. 2. Example of evolution of the test loss during training of the AlexNet CNN model pruned with 0.5 and 0.75 factor. Lines represent the expected loss, while shaded areas are delimited by the loss 1st and 99th percentiles.

quality distribution;

• In light of the problem complexity, we propose a solution strategy called DepL, allowing to achieve the target learning quality distribution at the minimum cost;

• We analyze DepL's complexity and competitive ratio, proving the former to be polynomial and the latter to be both low and constant;

• We show that DepL closely matches the optimum and consistently outperforms the state of the art.

The rest of the paper is organized as follows. Sec. II further motivates our approach, considering a real-world example and showing why solely focusing on the expected learning quality may be suboptimal. Sec. III formally introduces our system model and problem formulation. The DepL solution is then presented in Sec. IV and analyzed in Sec. V. Finally, after evaluating DepL's performance in Sec. VI and discussing related work in Sec. VII, we conclude the paper in Sec. VIII.

II. The Importance of Dependable Training

We now illustrate the reasons why it is important to take a dependable approach to ML model training. As an example, assume that the popular AlexNet CNN model [7] for image classification has to be trained at the network edge, using the CIFAR-10 dataset [8]. To make it more amenable to the limited computational capability of edge servers, the model can be compressed by pruning it with a given factor. Clearly, the larger the pruning factor, the smaller and the less complex the model becomes, thus leading not only to a less demanding model for inference but also to a lower cost during training. On the other hand, a more compressed version of the model exhibits higher sensitivity to a drift in the distribution of the input data and may provide less accurate output during inference due to the reduced number of model weights [9], [10]. *An aspect that however has been widely overlooked so far is how the dependability of training, i.e., the confidence level in the accuracy achieved through training, may be severely impacted by the version of the model and the dataset that are used. Importantly, these, on their turn, depend upon the computational capability of the nodes executing the training as well as the data sources used for the training.*

An example of experimental evidence of the above effect is given by the results shown in Fig. 2, obtained using two different versions of the AlexNet model, one with pruning factor F_1=0.5 and the other with pruning factor F_2=0.75, and adopting the gradient descent optimizer with learning rate and momentum, respectively, equal to 10^{-3} and 0.9, and batch size equal to 64. As a proxy metric for the achieved accuracy, the figure presents the evolution of the test loss through the training epochs. Three important facts can be noted:

1) The trajectories of the test loss evolution over the epochs are not deterministic but stochastic in nature, as highlighted by the shaded areas;
2) Their behavior visibly depends on the model that is used;
3) Additionally, the average and the empirical distribution of the loss values may exhibit a different trend, since, e.g., as the average gets smaller, the variance may instead increase (as in the case of F_2, blue curve).

Thus, upon establishing learning strategies for a given ML task, it is critical to account for all relevant aspects, including the dependability of the trained models, besides the training computational cost or the robustness of the compressed model. To properly tackle training dependability, in the following we develop a system model accounting for the stochastic behavior of the test loss evolution, and envision a solution strategy that selects the best learning strategy to achieve a target accuracy *with a target probability*.

III. System Model and Problem Formulation

System model. We tackle a distributed learning scenario with a set of network nodes, including cloud, edge, and far-edge nodes, that can contribute to a learning task, and a set of data sources that can transfer data to them. All such nodes are assisted by a learning orchestrator located at an edge server, which, given an ML task at hand, selects (i) the data to be leveraged for the model training, (ii) the ML model to be used, as well as (iii) the set of nodes, hereinafter referred to as learning nodes, that should contribute to the training of such a model. To encompass a wide range of distributed learning paradigms, including federated learning (FL) and sequential learning, we consider that the orchestrator identifies clusters of learning nodes, with each cluster performing the vanilla FL [11], and, possibly, contributing to the training process for a given number of epochs before handing the model over to the next cluster. Additionally, each cluster may use a different variation of the ML model (i.e., a full or a compressed version). The scenario thus includes the following main elements:

• Clusters of learning nodes, $n \in \mathcal{N}$, that, for simplicity and without loss of generality, are assumed to include homogeneous nodes with equal communication and computational capability. We thus denote with $r(n)$ the computational capability of the generic node in cluster n and with $b(n)$ the bandwidth available on the link between the FL coordinator within cluster n and each learning node. Each unit of computing resource has a *cost*, $\kappa(n)$, which depends on the equipment available at the cluster nodes;

• Datasets, $d \in \mathcal{D}$, each generated by a data source, that can be conveyed to node clusters. Let $\beta(d,n)$ be the total network cost to transfer dataset d from the source to the nodes[1] of cluster n before the cluster starts training the model;

• ML models suitable for the task at hand, in their full or compressed versions. They are represented as elements of the models set, $m \in \mathcal{M}$. Each model is associated with a size $s(m)$ and a complexity factor. It follows that, at each epoch, the communication latency within cluster n due to the learning nodes sending their updated model m to their FL coordinator is $\frac{s(m)}{b(n)}$. Also, given the model complexity, let $c(k,m,d)$ indicate the quantity of computational resources needed by a learning node to train an epoch k of model m using data from dataset d.

Given a model $m \in \mathcal{M}$ and being cluster $n \in \mathcal{N}$ the one currently training m with dataset $d \subseteq \mathcal{D}$, we define as $l^{\text{run}}(k,d,m)$ the global change in the loss achieved by training model m over d for the k-th epoch. Notice that training aims at minimizing the loss, hence, l^{run} will be negative if the training does progress. Instead, switching from model $m(k-1)$ to model $m(k)$ changes the loss by $l^{\text{sw}}(k,d,m(k-1),m(k))$, with k being the current epoch and $d \subseteq \mathcal{D}$ the dataset being used. Notice that l^{sw} is typically positive, as model switching often results – in the short term – in an increase of the loss. Importantly, all these values are *input* information to our problem; as discussed in Sec. VI-A, estimating them is an orthogonal problem to our own.

Decisions and problem formulation. The orchestrator has to make the following decisions:

• Whether cluster $n \in \mathcal{N}$ contributes to the learning at epoch k or not, expressed through binary variable $y(k,n) \in \{0,1\}$;

• Whether cluster $n \in \mathcal{N}$ uses dataset $d \in \mathcal{D}$ for learning, expressed through binary variable $z(d,n) \in \{0,1\}$;

• The amount of resources allocated for the model training by a node of a selected cluster, expressed through the real variable $x(k,n) \leq y(k,n) r(n)$;

• The total number K of epochs to run;

• For each epoch, the model version $m(k) \in \mathcal{M}$ to use.

Given the above decisions and neglecting the one-time data transfer from sources to clusters, the duration of epoch k is given by the computation and communication time of the selected cluster, i.e.,

$$T(k) = \frac{c(k,d,m(k))}{x(k,n)} + \frac{s(m(k))}{b(n)} \,. \tag{1}$$

Concerning the test loss $\ell(k)$, it can be computed recursively by accounting for the two l-contributions, i.e.,

$$\ell(k) = \ell(k-1) + l^{\text{run}}(k,d,m(k)) + l^{\text{sw}}(k,d,m(k-1),m(k)) \,.$$

where $\ell(0)$ is the initial loss value. Note how the last term in the above equation reduces to zero if no model switching occurs from epoch $k-1$ to epoch k.

[1]Again, for simplicity and without loss of generality, we consider that, given d, each cluster node receives an equal portion of the dataset.

The objective of the orchestrator is to minimize the total cost, subject to the fact that the learning quality target is achieved within the target time:

$$\min_{x,y,z,K,m} \quad \sum_{k=1}^{K} \sum_{n \in \mathcal{N}} \left(\sum_{d \in \mathcal{D}} \beta(d,n) z(d,n) + \kappa(n) x(k,n) \right) \tag{2}$$

$$\text{s.t.} \quad \ell_\omega(K) \leq \ell_\omega^{\max} \tag{3}$$

$$\textstyle\sum_{k=1}^{K} T(k) \leq T^{\max} \tag{4}$$

$$x(k,n) \leq y(k,n) r(n) \quad \forall k,n \tag{5}$$

$$z(d,n) \leq y(k,n) \quad \forall d,n,k. \tag{6}$$

It is important to highlight that in (3) $\omega \in [0,1]$ indicates a specific *quantile* of the loss, as opposed to a loss expected value. By tweaking ω, we are therefore able to enforce different levels of risk, depending upon the scenario and application at hand. Also, notice how (6) implies that inactive nodes neither receive nor use datasets.

As proven in Property 1 later, directly optimizing the problem above is prohibitively complex, as the problem is NP-hard. Accordingly, to efficiently and effectively solve large-scale problem instances, we propose a heuristic solution named *Dependable Learning (DepL)*, as described below.

IV. The DepL Solution

The DepL solution effectively tackles two critical issues of the system under study. First, the need for dependable training, seeking to keep a target quantile ω of the loss distribution below a threshold value $\ell_\omega^{\max}$ that is deemed to be acceptable for the application at hand. Second, the complexity of the decisions to make, which, along with the sheer amount of existing options to explore, makes finding the optimal solution impractical for realistic-sized problem instances. To address the first issue, DepL makes three main decisions:

1) selecting the *datasets* to use for learning;
2) choosing the *model* to use at each epoch, switching models as needed;
3) identifying the cluster of *learning nodes* to call for at each epoch, and the resources they should commit.

Then, to tackle the second issue, DepL efficiently makes orchestration decisions by *decoupling* them, while *iterating* as needed so as to account for their mutual influence.

To this end, it is critical to establish the order in which decisions are made. DepL properly accounts for the decisions' mutual influence by making decisions that strongly influence each other sequentially. We observe that the mutual influence is strongest between (i) dataset and model selection, since different models may call for different data, and (ii) model selection and resource allocation, as insufficient resources may cripple the model performance. Additionally, DepL seeks to make first decisions with the deepest impact on the overall training performance. To do so, we remark that:

• Dataset selection is related to the model selection *and* affects the resources needed at the nodes, as more data invariably means more processing at the nodes;

• Model selection is less critical, yet very relevant as it impacts both the learning progress (hence, how many epochs

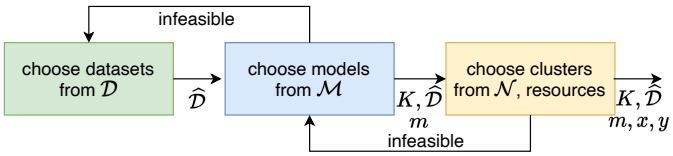

Fig. 3. The main steps of the DepL solution strategy.

we need) and the required computational resources (hence, how long each epoch will take);

• The impact of the resources and nodes to use is limited to the epoch duration and cost.

Based on the above, DepL takes the approach depicted in Fig. 3, making dataset selection decisions in the outermost loop, then model selection, and finally resource allocation. Importantly, a further reason to make decisions in the order highlighted in Fig. 3 has to do with how difficult it is to *estimate* their consequences. Indeed, choosing the data sources to use for learning requires sophisticate statistics [12] and feedback loops [13]. The impact of model selection on the overall learning is somehow better understood, though with significant uncertainty [4], and finding the right resources to run a given model can be mapped to many well-studied problems [14], [15]. By *making harder-to-estimate decisions less often, we can reduce the impact of estimation errors* on the overall solution quality.

Furthermore, in view of the above observations, we make each decision by considering the *least* restrictive option for the subsequent ones. As an example, while choosing the data, we assume that the most complex (i.e., full) version of the model is chosen in the subsequent step, and then that the cluster with the largest amount of available resources is used and they are fully allocated to the model training. Indeed, since adding more data is the most consequence-fraught decision to make, we want to avoid doing that, unless it is utterly necessary; similarly, a simpler model is used only if the available computational and networking resources are insufficient to make a more complex model work.

A. Dataset selection

Adding new datasets to improve learning quality may have undesirable consequences: (i) adding a new dataset implies that more training should be performed (hence, more resources are needed at the nodes using the dataset); (ii) more datasets typically imply more network resources to consume for their collection *and*, hence, a larger network. Accordingly, we begin with no selected datasets and add them as needed. Importantly, at this stage, we take the learning time as proxy of the overall training cost, as we do not directly tackle resource allocation.

Let us denote the set of datasets selected for learning with $\widehat{\mathcal{D}} = \{d{\in}\mathcal{D}\colon\ \exists\, n{\in}\mathcal{N}\colon\ z(d,n){=}1\}$. Let also $\widehat{K}(\widehat{\mathcal{D}})$ denote the number of epochs needed to reach convergence if the datasets in $\widehat{\mathcal{D}}$ are used, and by $\widehat{T}(\widehat{\mathcal{D}})$ the duration of each epoch. Notice that considering at this stage a fixed resource allocation implies that the epoch duration depends only on the selected datasets.

We remark that adding a new dataset to $\widehat{\mathcal{D}}$ has two contrasting effects on the training epochs. On the one hand, more learning is performed at each epoch – hence, fewer epochs are needed to reach the target learning quality. On the other hand, more data is collected and processed at each epoch – hence, each epoch lasts longer. Our key observation is that these two effects do not have the same evolution: (i) Adding more data decreases $\widehat{K}(\widehat{\mathcal{D}})$) according to a logarithm law [16], [17]; (ii) Processing and time to transmit model updates, $\widehat{T}(\widehat{\mathcal{D}})$ grow instead linearly with the quantity of data. It follows that, intuitively, if adding a new dataset does not help (specifically, it does not make convergence faster), adding another one will *also* not help. More formally, $\widehat{K}(\widehat{\mathcal{D}})\widehat{T}(\widehat{\mathcal{D}})$ is *submodular*.

Recalling that here we take the learning time as proxy of the cost, and that the only decision we have to make is selecting the datasets, we write the problem to solve as:

$$\min_{\widehat{\mathcal{D}}\subseteq\mathcal{D}} \widehat{K}(\widehat{\mathcal{D}})\widehat{T}(\widehat{\mathcal{D}}), \tag{7}$$

which is an *unbounded submodular minimization problem* [18]. Such problems can be solved to optimality in polynomial time via several different algorithms. Examples range from the traditional ellipsoid method [18] (based on the notion of Lovász extension [18], linking submodular combinatorial problems to convex continuous ones) to more modern approaches [19] that achieve *strongly polynomial* runtime, only depending upon the problem size.

In summary, regardless of the concrete algorithm used to solve (7), it is possible to choose the datasets in such a way that the learning time (hence, cost) is minimized. This means that DepL solves to optimality the first step of the solution strategy, although not original problem in (2), which is NP-hard (as per Property 1).

B. Model selection

The high-level goal is now to select a model in $\mathcal{M}$ to use at each epoch. To cope with the complexity and the combinatorial nature of the decision to make, we leverage an *expanded graph* approach, predicated upon:

1) Creating a graph whose vertices represent the possible states of the system;
2) Adding edges therein representing the possible decisions and their effects;
3) Seeking the path (hence, a sequence of decisions) that connects the current state with a feasible one and has the lowest weight (i.e., cost).

As mentioned above, at this stage the datasets to use are given, and we assume that the most capable node cluster, and all the resources therein, can be used.

To build the expanded graph, let η be an integer parameter determining the accuracy with which we discretize learning time and test loss. Each vertex of the graph is then associated with a 4-uple $(m(k), k, T(k), \ell_\omega(k))$, where:

- $m(k)$ is a model in $\mathcal{M}$;
- k is an epoch in $1, \ldots, K$;
- $T(k){=}\frac{i}{\eta}T^{\max}$, with $i{=}1, \ldots, \eta$, is the elapsed learning time in the state represented by the vertex;

- $\ell_\omega(k)=\frac{j}{\eta}\ell(0)$, with $j=1,\ldots,\eta$ and $\ell(0)$ being the initial loss value, reflects, in a similar way, the ω-quantile of the current loss.

The total number of vertices is then $|\mathcal{M}|K\eta^2$. Such vertices represent *a subset* of all possible states of the system (e.g., there is no vertex with $T(k)=\frac{0.5}{\eta}T^{\max}$), while their indices keep track of the corresponding elapsed training time. This allows the size of the graph to be manageable, at the expense of precision.

Concerning edges, we add an edge from vertex $(m(k),k,T(k),\ell_\omega(k))$ to vertex $(m(k+1),k+1,T(k+1),\ell_\omega(k+1))$ if (i) it is possible to switch from $m(k)$ to $m(k+1)$, (ii) it takes at most $T(k+1)-T(k)$, and (iii) the loss changes by at most $\ell_\omega(k+1)-\ell_\omega(k)$. The weight of such edge represents the cost associated with the switch. Edges are also created between vertices corresponding to the same model, i.e., indicating no model switch.

Finally, we add a virtual vertex Ω and a zero-weight edge from any vertex representing a feasible solution (i.e., with an elapsed time lower than $T^{\max}$ and a loss quantile lower than $\ell_\omega^{\max}$) to Ω. Since vertices connected to Ω correspond to feasible states, and edges in the graph are a conservative representation of the outcome of model-switching decisions, any path connecting the current state to Ω corresponds to a feasible set of decisions. The lowest-weight among such paths represents then the lowest-cost, i.e., *optimal*, decisions we can make, given the graph representation. Such decisions may not be optimal solutions to the original problem, due to the fact that not all states are represented by vertices in the graph. However, by increasing η, we can reduce this issue and get arbitrarily close to the optimum, at the cost if increasing the size of the graph and the solution complexity.

There is an additional problem to deal with, stemming from the fact that expanded-graph approaches work with *additive* constraints [20], [21], i.e., constraints where the effect of a sequence of decisions (in our case, model selection at different epochs) can be expressed as the sum of the effects of individual decisions (in our case, the single l^{run} components). Such a property is required in order to ensure that the features of a path on the expanded graph (i.e., the associated learning time or loss quantile) can be expressed as the sum of features of individual edges therein. The property holds for the elapsed time, and it would be true for the expected loss, but not for the loss quantile. Indeed, we know the effects of model selection decisions as distributions, e.g., through their probability density function (pdf); however, the pdf of the sum of two random variables is not the sum of the individual pdfs, but their convolution.

We face this hurdle by adopting the following approach to deal with the evolution of the value of loss-quantile, based on function transforms:

1) We move to the transformed domain, replacing pdfs with their Laplace transforms;
2) Convolutions between pdfs are thus replaced by products of transformed pdfs;
3) We further compute the logarithm of the transformed pdfs, so that products can be computed as sums.

It is important to highlight that the aforementioned approach works in both cases (i) when pdfs of the test loss are known exactly (in which case the transforms and logarithms can be computed symbolically), and (ii) when they are not (in which cases all operations are performed numerically). In the latter case, the fast Fourier transform (FFT) may be used *in lieu* of the Laplace transform, owing to the vast availability of fast, high-quality implementations.

Finally, if no model can be selected that meets the target learning time and quality, DepL goes back to the previous stage: the current dataset choice is blacklisted and the datasets selection is repeated, still considering the most complex model. Optimized model selection will then be attempted again under the new data selection.

C. Node and resource allocation

Once the datasets and models are chosen, the only remaining decisions to make concern: (i) the clusters to use for learning, and (ii) which resources each cluster has to commit. Importantly, neither decision affects how many epochs are needed for convergence, nor the distribution of the resulting loss value; on the other hand, clusters and resource selection have a very strong influence on the learning time (i.e., the duration of each epoch) and the resulting cost.

We make the key observation that the resulting problem – including its decisions and their effects – can be mapped, one-to-one, to the classic problem of VNF placement, also referred to as service embedding. Specifically: (i) clusters are mapped to servers, which can run VNFs; (ii) models are mapped to the VNF themselves; (iii) model complexity is mapped to VNF requirements. As a consequence, we can use any VNF placement strategy to solve this stage of our own problem. VNF placement is a very well studied problem, with several efficient and effective schemes existing for it. In particular, we take [22] as a reference, and obtain the same $O(1+\varepsilon)$ competitive ratio and $O(1/\varepsilon)$ complexity.

Again, in the case where no cluster or resource allocation can be found that can make the model be trained within the target time, DepL goes back to the previous decision stage, i.e., model switching, again blacklisting the model choices that have proven insufficient and considering that any cluster and resource can be used. The overall process depicted in Fig. 3 may therefore be iterated till a feasible solution is found, or the existing options have been exhausted.

V. Problem and Solution Analysis

In the following, we formally prove several important properties about the problem we solve and our DepL solution. Our results can be summarized as follows:

- The problem we solve is NP-hard (Property 1), hence, a heuristic solution is required;
- The DepL solution has low, namely, polynomial computational complexity (Property 2);

- The solutions yielded by DepL are provably very close to the optimum (Property 3).

Beginning from the problem, we prove its NP-hardness via a reduction from the knapsack problem.

Property 1. *The problem of optimizing (2) subject to constraints (3)–(6) is NP-hard.*

Proof: We prove NP-hardness by reduction from the knapsack problem, which is known [23] to be NP-hard. To do so, we provide a polynomial-time procedure transforming any instance of the knapsack problem into an instance of our problem. Recall that the input to the knapsack problem consists of a set $\mathcal{I}=\{i\}$ of items, each with weight w_i and value v_i, along with a maximum weight $w^{\max}$. We have to decide whether to take each item i with the goal of maximizing the total value, subject to the constraint that the total weight does not exceed $w^{\max}$. Given the above, we can create an instance of our problem where: (1) items correspond to models in $\mathcal{M}$; (2) the learning improvement yielded by each model is deterministic and equal to the value of the corresponding item; (3) the requirements of each model are equal to the cost of the corresponding item; (4) there is only one dataset and only one cluster, with sufficient capabilities to run any model; (5) all data transfer costs $\beta(d,n)$ are set to zero. It can be seen by inspection that reduction is polynomial in complexity – indeed, constant, as it has no loops. As we have successfully reduced an instance of an NP-hard problem to an instance of our own, we can conclude that our problem is NP-hard. ■

It is also worth remarking that the instance of our problem generated by the reduction is a very simple one; this suggests that our problem is significantly more complex than the knapsack problem in practice. It also implies that general-purpose heuristic approaches for the knapsack problem (or any other known problem) will work poorly with ours, rather, domain- and problem-specific strategies like DepL are warranted.

Concerning DepL, we can prove that it has a remarkably low, namely, quadratic worst-case computational complexity:

Property 2. *The DepL procedure has worst-case polynomial complexity of* $O\left(|\mathcal{D}|^2+|\mathcal{M}|^2K^2\frac{1}{\eta^4}+\frac{1}{\varepsilon}\right)$.

Proof: The total complexity is obtained by considering the three steps depicted in Fig. 3 and described in Sec. IV.

For dataset selection (Sec. IV-A), we leverage the results in [19], solving unbounded submodular minimization problems to the optimum with a computational complexity that is quadratic in the size of the set over which to optimize, i.e., $\mathcal{D}$. For model switching (Sec. IV-B), we have to compute the minimum-cost path over the expanded graph. The maximum number V of vertices in the expanded graph is given by all combinations of model, epoch, time, and loss, hence, $V \leq |\mathcal{M}|K\frac{T^{\max}}{\eta}\frac{\ell^{\max}}{\eta}$. Recalling that minimum-cost path algorithms, e.g., Dijkstra's, have a complexity of $O(V^2)$, it follows that the complexity of the model selection procedure is $O\left(|\mathcal{M}|^2K^2\frac{1}{\eta^4}\right)$. Finally, for cluster selection (Sec. IV-C), we rely on [22], which has a complexity of $O\left(\frac{1}{\varepsilon}\right)$, where ε is a configurable optimality parameter. Combining all contributions, we obtain $O\left(|\mathcal{D}|^2+|\mathcal{M}|^2K^2\frac{1}{\eta^4}+\frac{1}{\varepsilon}\right)$, which proves the thesis. ■

The last aspect we look at is the effectiveness of the decisions made by DepL. We can prove that the competitive ratio (i.e., the ratio of DepL's solution cost to the optimal one) is constant (i.e., it does not depend upon the problem size) and remarkably small.

Property 3. *The DepL solution has a constant competitive ratio, namely,* $\frac{1}{\eta}(1+\varepsilon)$.

Proof: Similarly to the proof of Property 2, we need to consider the competitive ratio of each of the steps described in Sec. IV, and then multiply them. Dataset selection (Sec. IV-A) is optimal, hence, the corresponding competitive ratio is 1. For model selection, instead, the only source of sub-optimality is the parameter η; specifically, as per [21, Theorem 4.3], the resulting competitive ratio is $\frac{1}{\eta}$. Finally, the approach in [22], which we consider as a reference for node selection, yields a competitive ratio of $1+\varepsilon$. ■

A constant competitive ratio like the one proven in Property 3 implies that the quality of the solutions found by DepL is not influenced by the size of the problem instance being solved; intuitively, DepL is *robust* to large problem sizes.

VI. Numerical Results

In this section, we first (Sec. VI-A) demonstrate how the l^{run} and l^{sw} parameters can be estimated through real-world experiments. Then (Sec. VI-B), we leverage the resulting data to compare the performance of DepL, the optimum, and the state-of-the-art.

A. Estimating the loss evolution

As observed in Sec. II, the evolution of the test loss over the training epochs, k, cannot be predicted with certainty, rather, a probabilistic representation thereof should be used. Previous work [24] has shown that the loss evolution can be well approximated by an inverse-square-root law, i.e., it is proportional to $\frac{1}{\sqrt{k}}$. We generalize such an expression as:

$$\ell^{\text{run}}(m,k) = \frac{A_m}{\sqrt{k+B_m}} \cdot x, \tag{8}$$

where $x \sim \mathcal{X}_m$ is a sample drawn from distribution $\mathcal{X}_m$, which is specific to the considered model and data. The parameters in (8) can be estimated as follows: (i) A_m and B_m are obtained through traditional parameter fitting techniques, e.g., least-squares, under the assumption that x=1; (ii) given A_m and B_m, $\mathcal{X}_m$ is obtained through distribution fitting.

Fig. 4(left) shows the empirical pdf of x for an example test loss curve, along with the fitted pdf $\mathcal{X}_m$, in the case where two clusters of nodes sequentially train the AlexNet CNN using the CIFAR-10 dataset. The two clusters are located, respectively, in the network edge and the far-edge, and perform FedAvg. The order in which the clusters contribute to the training goes from the highest (edge) to the lowest (far-edge) computing capability, as ultimately the model needs to be personalized

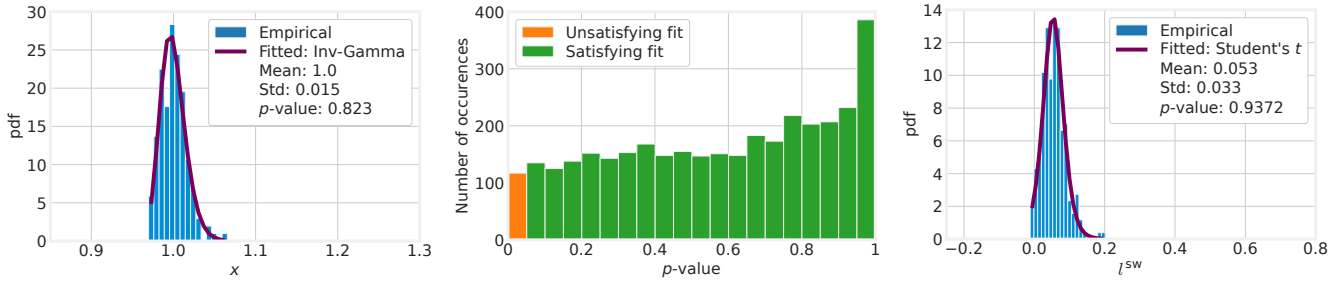

Fig. 4. Example distributions of $\mathcal{X}$ as defined in (8) along with the Inverse-Gamma fit (left); distribution of the p-values highlighting fitting quality (center); example distribution of l^{sw} as defined in (2) along with the Student's t fit (right).

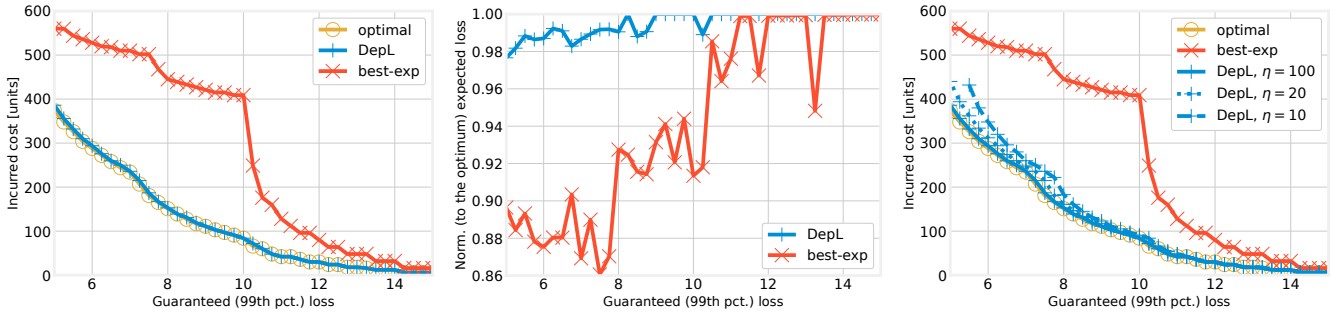

Fig. 5. AlexNet: performance of DepL and alternative benchmarks: cost (left); normalized expected loss (center); effect of η (right).

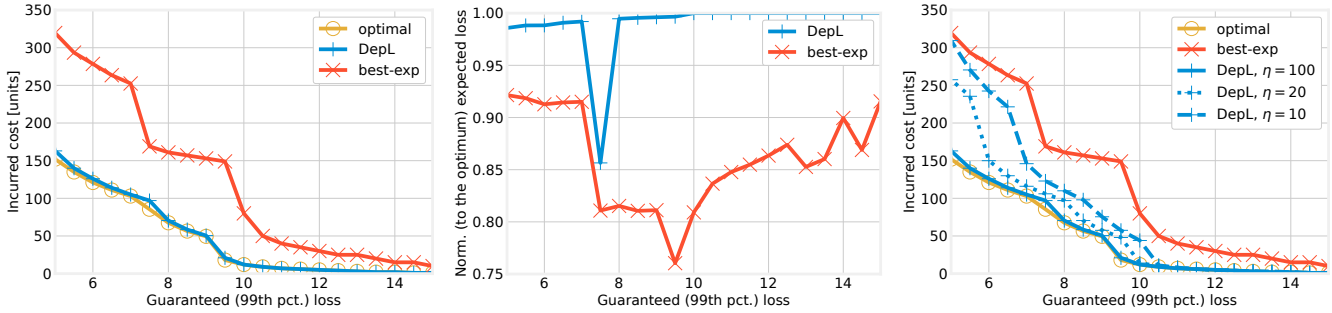

Fig. 6. MobileNet: performance of DepL and alternative benchmarks. Cost (left); normalized expected loss (center); effect of η (right).

with the data owned by IoT or user devices (which have indeed lower compute capabilities). At each training stage, the model is further pruned, so as to make it amenable for training at the lower-end nodes. Let K_1 (K_2) be the number of epochs performed at the first (second) stage, using a model pruned with factor F_1 (F_2); for Fig. 4(left) we set K_1=17, K_2=60, F_1=0.5, and F_2=0.75. The empirical pdf, and, more specifically, the standard deviation, highlight that, given a combination of learning parameters, the prediction of the next loss value exhibits significant uncertainty, thus further confirming the need for a probabilistic representation of the loss trajectory. Importantly, one can notice that the Inverse-Gamma distribution provides an excellent goodness of fit. The latter is measured by computing the p-value using the Kolmogorov-Smirnov (KS) test, with p-values higher than 0.05 corresponding to a good fit[2]. The histogram in Fig. 4(center) shows the p-values for the distributions obtained using all possible combinations of the number of epochs $K_1 \in [1, 20]$ and $K_2 \in \{1, 5, 15, 25, 40, 60\}$, and of the pruning factors $F_1, F_2 \in \{0.25, 0.36, 0.5, 0.75\}$: in almost all cases they are larger than the 0.05 threshold, again indicating the goodness of fit of the Inverse-Gamma distribution on the empirical distributions.

Model switching, e.g., compression, has a different effect: namely, a one-time increase in the loss (hence, a reduction in accuracy). We modeled this effect by l^{sw}, i.e., the difference between the test loss values after and before the model switching. Clearly, each combination of learning parameters yields a different distribution. Fig. 4(right) shows an example of l^{sw} distribution, representing the empirical and the fitted pdfs for F_1=0.36, F_2=0.5. The empirical pdf, in Fig. 4(right), shows that the support of l^{sw} distribution is composed of positive values, indicating an increase of the test loss after model switching. Again, such an increase cannot be predicted

[2]A p-value higher than 0.05 indicates that our null hypothesis, i.e., data are distributed according to an Inverse-Gamma distribution, cannot be rejected.

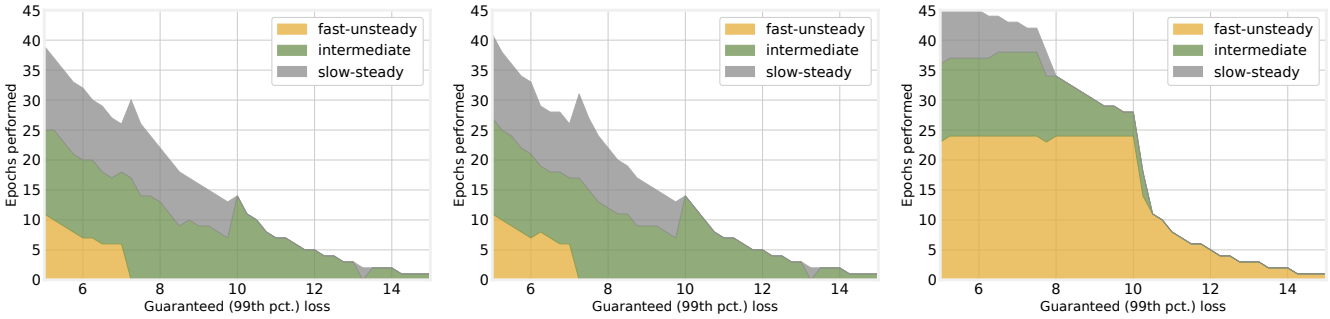

Fig. 7. Epochs spent with each model by the optimal solution (left), DepL (center), and *best-exp* (right).

with certainty; hence, we need a probabilistic representation of l^{sw}. In this case, the KS test reveals that satisfying p-values can be obtained using the Student's t-distribution.

Below, we use the above experimental estimation of the loss evolution to assess and benchmark the performance of DepL.

B. Performance evaluation

Since, as discussed in Sec. IV-A, the dataset selection is provably optimal, our performance evaluation focuses on model and node selection, i.e., steps 2 and 3 in Fig. 3. To this end, we benchmark our DepL solution described in Sec. IV-B against two alternatives:

- Optimal decisions, obtained by brute force (labelled as *optimal* in plots);
- A state-of-the-art solution, aiming to optimize the expected loss, inspired to [4] (labelled as *best-exp* in plots).

The best-exp scheme makes decisions of the same nature as our DepL; however, it accounts for the expected loss instead of any quantile thereof. Thus, comparing against best-exp is a valuable way to gauge the impact of accounting for the distribution of the loss besides its expected value. Finally, unless otherwise specified, for DepL we set η=200.

For all strategies, we consider as items of $\mathcal{M}$:

- either three versions of the AlexNet DNN pruned, respectively, with factors F_1=0.5, F_2=0.75, and F_3=0.9,
- or three versions of the MobileNet network, pruned according to the same factors.

Tab. I summarizes the features of the DNNs we use. Specifically, as the name assigned to the models suggests, the "fast-unsteady" model provides a fast learning on average, but it is liable to occasionally yield poor test loss performance; "slow-steady", on the other hand, provides slower but more reliable convergence, while "intermediate" yields balanced performance.

We consider a network infrastructure including two types of server (hence, clusters nodes in $\mathcal{N}$): A-class nodes, with capabilities set to $r(n)$=100 and processing cost $\kappa(n)$=1, and B-class nodes, with $r(n)$=50 and processing cost $\kappa(n)$=1.2. For simplicity, we assume that clusters contain one node each, and set $\beta(n,d) = 0$ for all nodes and sources. For all strategies, node assignment and resource allocation decisions are made following the methodology in [22].

TABLE I
ALEXNET AND MOBILENET VERSIONS USED FOR PERFORMANCE EVALUATION: EXPECTED AND VARIANCE OF LOSS IMPROVEMENT PER EPOCH, AND RESOURCE DEMAND

Model	Exp. loss improv.	Var. of loss imp.	Norm. needed resources	Exp. loss improv	Var. of loss imp.	Norm. needed resources
	AlexNet			MobileNet		
Fast-unsteady	1.02	0.031	2.6	1.02	0.029	1.8
Intermediate	1.00	0.010	1.4	0.95	0.012	1.0
Slow-steady	0.98	0.007	1	0.92	0.006	0.6

The first aspect in which we are interested is the relative performance of the strategies we compare, i.e., how good they are at minimizing the objective (2). The results for the AlexNet model are summarized in Fig. 5(left), showing the cost incurred for different target values of the loss 99th percentile. As one might expect, tighter loss requirements require more training, hence, a higher cost. Most interestingly, DepL outperforms best-exp and virtually matches the optimum. The difference is especially large when the requirements are *less* strict, hence, there is more margin for learning strategies that optimize cost over sheer performance.

Fig. 5(center), depicting the loss achieved by DepL and best-exp, normalized to the optimum, sheds additional light on this effect. All strategies meet the learning quantile constraint (3) with an equality sign, i.e., they provide exactly the required value of the loss 99th percentile. On the other hand, Fig. 5(center) shows that both best-exp and, to a much smaller extent, DepL yield a lower (i.e., better) expected loss than the optimum. Clearly, this is a waste of resources: since our constraint is on the 99th percentile of the loss, improving its expected value brings no benefit.

In Fig. 5(right), we assess the effect of decreasing the parameter η, which we use when building the expanded graph in Sec. IV-B, on the performance of DepL. We observe how there is indeed a significant margin to pursue different trade-offs between DepL's computational efficiency and the quality of the decisions it yields, as indicated by the analysis in Sec. V.

The same behavior can be observed from Fig. 6, presenting the behavior when MobileNet versions are used. We can observe that, compared to Fig. 5, costs tend to be lower, a testament to the better efficiency of MobileNet compared to AlexNet. Furthermore, costs decrease more steeply as the loss

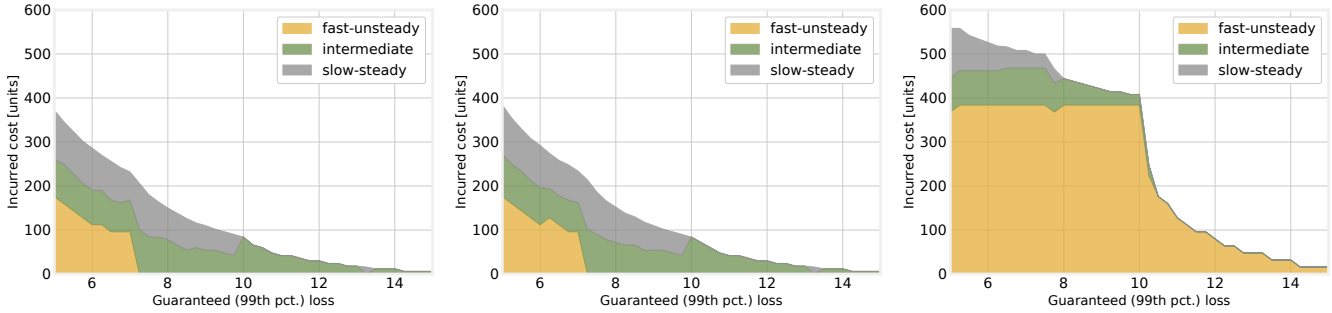

Fig. 8. AlexNet: cost incurred with each model by the optimal solution (left), DepL (center), and *best-exp* (right).

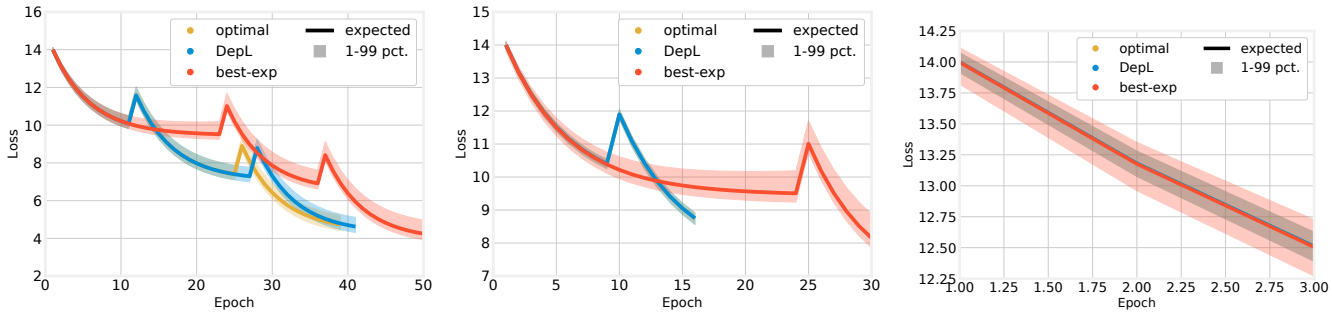

Fig. 9. AlexNet: evolution of the expected and guaranteed loss when the guaranteed loss is 5 (left), 9 (center), and 13 (right).

target increases, showing that MobileNet struggles more for more complex learning tasks. Besides these differences, it is important to remark how – exactly like in Fig. 5 – DepL significantly outperforms best-exp, and closely matches the optimum.

Moving back to AlexNet, in Fig. 7 we take a closer look at the decisions made by each strategy, specifically, at the number of epochs for which each of the models in Tab. I is used for training. DepL (center plot) makes very similar choices to the optimum (left plot), consistently with Fig. 5; in both cases, all models are used as required by the loss quantile target. Interestingly, when the target is loose, both strategies can only use the "intermediate" model, and avoid expensive model switches. On the contrary, as the target gets tighter, switches become necessary and all models are used. The right plot, depicting the decisions made by best-exp, shows a completely different picture. As best-exp optimizes the expected loss, it utilizes the "fast-unsteady" model as much as possible, which results in a higher number of epochs to run, hence, the higher costs presented in Fig. 5(left).

Fig. 8 shows the total cost of each strategy, broken down according to the model they use. As in Fig. 5(left), best-exp requires the most computational resources, hence, incurs the highest cost; also, in accordance with Tab. I, much of such cost is due to the "fast-unsteady" model. This confirms how solely focusing on the expected loss means performing more epochs (Fig. 7) *and* spending more in each of them (Fig. 8), while obtaining a needlessly low expected loss (Fig. 5(center)).

Finally, Fig. 9 shows the evolution of the loss across different epochs, for different strategies and values of the target learning quality; peaks correspond to model switches. In all plots, lines represent the expected loss, while the shaded area is delimited by the loss 1st and 99th percentiles. We can immediately observe that the lines corresponding to DepL (and the optimum) are shorter, i.e., fewer epochs are necessary to reach the target learning quality. Further, it is evident (especially in the middle plot) how best-exp is associated with wider shaded areas, i.e., a larger distance between the loss 1st and 99th percentiles. This is consistent with Fig. 7 and Fig. 8: from those figures, we can observe that best-exp tends to use more the "fast-unsteady" model, which is associated with a larger variance (as per Tab. I). Furthermore, Fig. 9 confirms how DepL virtually always matches the optimum, making model switching decisions that account for both the expected value *and the distribution* of the loss.

VII. Related work

DepL is related to three main research areas: resource-aware distributed DNN training, DNN model compression, and robust DNN training. In the former case, the goal is to locate and exploit the best resources for a given training task. The main decision is often choosing the nodes to involve in training, based on their speed [1], [2], the quantity [1], [25] and quality [3] of their data, the speed and reliability of their network [2], [26], as well as trust [27]. In all cases, the model to train is assumed to be given and fixed. Recent works [28] target more complex scenarios, where layers of the DNN can be duplicated at different learning nodes if needed. The studies in [4], [29] also seek to adapt the learning task to the available resources, choosing from different existing models.

Model compression is a set of techniques to transfer the knowledge from a model to a different (usually simpler) one. Prominent among such techniques is model pruning, which consists in removing some weights from a DNN model,

assuming that very few of the parameters of a model actually impact its performance. Choosing the right parameters to prune is a challenging problem itself; many solutions adopt an iterative approach [30], by observing the evolution of the parameter values. Other compression techniques include knowledge distillation [31], where a "student" model is trained to mimic the decisions of a "teacher" model. Later studies have tackled how distillation can be combined with generative models [32] and domain adaptation [31].

Robustness in distributed DNN training is an emerging topic. Most works set in a FL scenario and focus on node selection. Such a robustness objective is balanced against fairness consideration in [33], aiming at distributing the training load as evenly as possible. Robustness considerations are also combined with communication efficiency in [34] and data quality. Importantly, all the aforementioned works deal with node selection; the reliability of the DNN model and the training process are never accounted for.

An alternative approach to adding reliability to DNN results is the so-called *conformal prediction* [5], [6]. Conformal approaches create a set of possible results of a given DNN (e.g., predicted class) such that the correct result (e.g., the true class) lies in the set with a target probability; intuitively, better models will have smaller conformal sets. Conformal prediction techniques are applied *after* training, hence, they can be seamlessly combined with DepL, and indeed benefit from the high-quality training it yields.

VIII. Conclusions

We presented, analyzed and evaluated DepL, a strategy for the dependable training of distributed ML models. Unlike previous work, DepL can guarantee that the target learning quality (e.g., accuracy) is reached *with a target probability*. DepL achieves this goal by making joint, high-quality decisions about the data, nodes, and models to use, including the time at which model switching (through, e.g., compression) shall be performed. We have formally proved that DepL achieves near-optimal decisions with a constant competitive ratio and low complexity. We have further shown that DepL closely matches the optimum and significantly outperforms the state-of-the-art.

Acknowledgment

This work was supported by the SNS-JU-2022 project ADROIT6G under the European Union's Horizon Europe research and innovation programme under Grand Agreement No. 101095363.

References

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

[2] 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 JSAC*, 2019.

[3] H. Wu and P. Wang, "Fast-convergent federated learning with adaptive weighting," *IEEE TCCN*, 2021.

[4] 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.

[5] A. N. Angelopoulos, S. Bates, A. Fisch, L. Lei, and T. Schuster, "Conformal risk control," *arXiv preprint arXiv:2208.02814*, 2022.

[6] M. Zecchin, S. Park, and O. Simeone, "Forking uncertainties: Reliable prediction and model predictive control with sequence models via conformal risk control," *arXiv preprint arXiv:2310.10299*, 2023.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

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

[9] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *Proceedings of machine learning and systems*, vol. 2, pp. 129–146, 2020.

[10] L. Liebenwein, C. Baykal, B. Carter, D. Gifford, and D. Rus, "Lost in pruning: The effects of pruning neural networks beyond test accuracy," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 93–138, 2021.

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

[12] 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.

[13] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *IEEE INFOCOM*, 2020.

[14] G. Sallam and B. Ji, "Joint placement and allocation of virtual network functions with budget and capacity constraints," in *IEEE INFOCOM*. IEEE, 2019.

[15] D. Harris and D. Raz, "Dynamic vnf placement in 5g edge nodes," in *IEEE NetSoft*, 2022.

[16] T. Linjordet and K. Balog, "Impact of training dataset size on neural answer selection models," in *ECIR*, 2019.

[17] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *IEEE ICCV*, 2017.

[18] L. Lovász, "Submodular functions and convexity," *Mathematical Programming The State of the Art: Bonn 1982*, pp. 235–257, 1983.

[19] T. Itoko *et al.*, "Computational geometric approach to submodular function minimization for multiclass queueing systems," in *IPCO*, 2007.

[20] J. Martín-Pérez *et al.*, "KPI guarantees in network slicing," *IEEE/ACM Trans. on Networking*, 2021.

[21] G. Xue *et al.*, "Finding a path subject to many additive qos constraints," *IEEE/ACM Trans. on networking*, 2007.

[22] H. Feng *et al.*, "Approximation algorithms for the nfv service distribution problem," in *IEEE INFOCOM*, 2017.

[23] H. Kellerer, U. Pferschy, D. Pisinger, H. Kellerer, U. Pferschy, and D. Pisinger, "Introduction to np-completeness of knapsack problems," *Knapsack problems*, 2004.

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

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

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

[27] A. Imteaj *et al.*, "Fedar: Activity and resource-aware federated learning model for distributed mobile robots," in *IEEE ICMLA*, 2020.

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

[29] F. Paissan *et al.*, "Scalable neural architectures for end-to-end environmental sound classification," in *IEEE ICASSP*, 2022.

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

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

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

[33] T. Li, S. Hu, A. Beirami, and V. Smith, "Ditto: Fair and robust federated learning through personalization," in *ICML*, 2021.

[34] F. Ang, L. Chen, N. Zhao, Y. Chen, W. Wang, and F. R. Yu, "Robust federated learning with noisy communication," *IEEE Transactions on Communications*, 2020.