Achieving Machine Learning Dependability Through Model Switching and Compression

# Achieving Machine Learning Dependability Through Model Switching and Compression

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

***Abstract*—Machine learning (ML) can be often *distributed*, owing to the need to harness more resources and/or to preserve privacy. Accordingly, distributed learning has received significant attention from the literature; however, most works focus on the *expected* learning quality (e.g., loss) attained and do not consider the *distribution* thereof. It follows that ML models are not dependable, and may fall short of the required performance in many real-world cases. In this work, we tackle this challenge and propose DepL, a framework attaining *dependable learning orchestration*. DepL efficiently makes joint, near-optimal decisions concerning (i) which data to use for learning, (ii) the ML models to use – chosen within a set of full-size models and compressed versions thereof – and when to switch from one model to another, and (iii) the clusters of physical nodes to use for the learning. DepL improves over previous works by guaranteeing that the learning quality target (e.g., a minimum loss) is achieved *with a target probability*, while minimizing the learning (e.g., energy) cost. DepL has provably low polynomial computational complexity and a constant competitive ratio. Further, experimental results using the CIFAR-10 and GTSRB datasets show that it consistently matches the optimum and outperforms state-of-the-art approaches (30% faster learning and 40–80% lower cost).**

***Index Terms*—Distributed learning, network support to machine learning, dependable learning, learning guarantees**

## I. Introduction

Two main trends can be observed in the field of machine learning (ML) models and, especially, deep neural networks (DNNs): on the one hand, their capabilities tend to increase; on the other hand, they require an increasing amount of data and resources for the learning process. To cope with such a shortcoming, two approaches have emerged, to wit, distributed learning and model compression. Distributed learning allows the use of resources and data available at multiple nodes to obtain a faster learning process in a privacy-preserving manner. Model compression, instead, includes a set of related techniques (from model pruning to knowledge distillation) whose high-level purpose is making a simpler model perform like a more complex one, thus *transferring* the knowledge from the latter to the former.

Several approaches appeared in the literature have variously combined distributed learning and model compression. As an example, [1], [2] adapt the resources used for learning.

F. Malandrino and C. F. Chiasserini are with CNR-IEIIT and CNIT, Italy. G. Di Giacomo and C. F. Chiasserini are with Politecnico di Torino, Italy. C. F. Chiasserini is also with Chalmers University of Technology, Sweden. M. Levorato is with University of California, Irvine, USA. This work was supported by the SNS JU under the EU HE programme under Grant Agreement No. 101192521 (MultiX project).

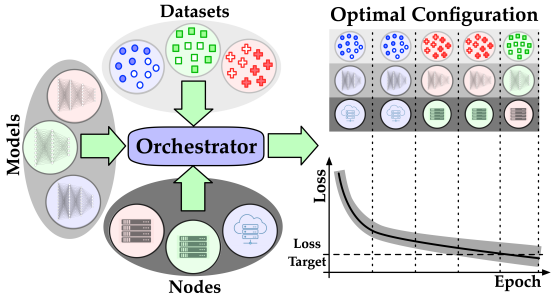


Fig. 1. DepL enables the optimization of dependable, distributed learning of neural networks over heterogeneous nodes, datasets, and model architectures. The optimal configuration is designed to control the uncertainty in the loss progression along learning epochs.

Following an opposite approach, [3], [4] choose instead the ML model to use by accounting for the available resources, with some recent works [5] performing a mutual adaptation between models and resources. In all cases, the focus is on the average or expected learning quality (e.g., loss [5]) achieved through the selected model and resources. However, such an approach might be unsuitable for modern mission-critical, e.g., safety-related applications, which require *dependable* learning for DNNs.

Dependable learning for DNNs is indeed the high-level purpose of our work. Specifically, we seek to attain a given learning quality target, namely, a maximum loss value or a minimum accuracy level, within a target time and – crucially – with at least a *target probability*, whilst minimizing the learning cost. We achieve this goal by tackling the main factors affecting learning dependability in the relevant case where the fine-tuning of pre-trained models (either full-size models or compressed versions thereof) is necessary [6]–[8]. Such factors include the *data* and the *model* (full-size or compressed version) to be used, and the *resources* to allocate to the learning process. Our solution concept, called DepL for *dependable learning*, accounts for all the above contributions in a holistic fashion, thus making efficient and effective decisions. Notice how, although in this work we choose from different compressed versions of the same model, both the problem we consider and our DepL solution work unmodified for arbitrary models.

As reference scenario, we consider a multi-tiered network (e.g., far edge, edge, and cloud) where nodes at different tiers can cooperate to fine-tune a DNN model, with the support of a *learning orchestrator*. DepL runs at the orchestrator itself and makes joint decisions about the three main aspects of the

learning process:

1) the data sources to use for the learning process;
2) the DNN models to fine-tune, as well as when (at which epoch) to switch among them;
3) the node clusters to leverage for learning and the resources they shall commit.

Such decisions interact with one another in a complex (and, often, counterintuitive) fashion; accordingly, DepL accounts for such interactions and the resulting effect on the *distribution* of the learning quality, in addition to the expected value thereof. As shown later in the paper, DepL's decisions are near-optimal and minimize the total learning cost; at the same time, DepL has low (namely, polynomial) worst-case computational complexity.

Our main contributions can thus be summarized as follows:

- We devise a synthetic, yet comprehensive, system model, describing how different aspects of distributed learning and decisions therein impact the learning quality distribution;
- We analyze the problem complexity, and find it too high to directly solve the problem to optimality;
- Accordingly, we propose our novel solution strategy called DepL, minimizing the learning cost subject to constraints on the learning time and the learning quality distribution;
- We prove that DepL has polynomial complexity and constant competitive ratio;
- We evaluate the performance of DepL using state-of-the-art, *de facto* standard DNNs and wo datasets, i.e., the classic CIFAR-10 benchmark and the real-world road sign recognition dataset GTSRB. Our results demonstrate that DepL performs remarkably close to the optimum and substantially better than existing approaches. Specifically, relatively to its alternatives, it can reduce learning cost by 40–80%, depending on the required loss guarantees, while reducing the number of learning epochs by 30%.

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. Dependable Learning

We now illustrate the reasons why it is important to take a dependable approach to learning (Sec. II-A), and highlight the difference with respect to the approach of conformal prediction and neural architecture search (Sec. II-B).

### A. The Importance of dependable learning

As an example, consider that the popular AlexNet CNN model [9] for image classification has to be fine-tuned at the network edge, using the CIFAR-10 dataset [10]. To make it more amenable to the limited computational capability of edge servers a pre-trained model can be compressed, by pruning it with a given factor, prior to its fine-tuning using local data. 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 the learning process. 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 [11]–[13]. *An aspect that however has been widely overlooked so far is how the dependability of learning, i.e., the confidence level in the accuracy achieved, may be severely impacted by the version of the model and the dataset that are used. Importantly, these, in their turn, depend upon the computational capability of the nodes executing the learning as well as the data sources that are used.*

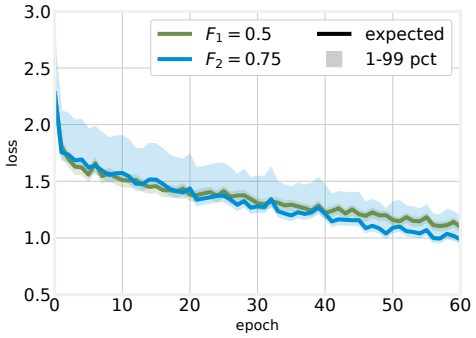


Fig. 2. Example of evolution of the testing loss during learning in the case of the AlexNet CNN model pruned with 0.5 and 0.75 factor. Lines represent the expected loss, while shaded areas are delimited by the loss 1st and 99th percentiles, measured over a total of 1,000 runs.

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. We ran a total of 1,000 runs, with different seeds. As a proxy metric for the achieved accuracy, the figure presents the evolution of the testing loss through the learning epochs. Three important facts can be noted:

1) The trajectories of the testing 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).

An intuitive explanation of the diverging behavior between average and variance that we observe in Fig. 2 is given by the fact that, when there are many DNN parameters but relatively few samples to learn from, the effect of random initialization and random SGD decisions becomes more significant.

Thus, upon establishing learning strategies for a given ML task, it is critical to account for all relevant aspects, including

the dependability of the DNN models, besides the learning computational cost or the robustness of the compressed model. To properly tackle learning dependability, in the following we develop a system model accounting for the stochastic behavior of the testing loss evolution, and envision a solution strategy that selects the best learning strategy to achieve a target accuracy *with a target probability*.

### *B. Relationship to conformal prediction and neural architecture search*

Conformal prediction [14] is an emerging ML framework aiming at achieving arbitrary, guaranteed accuracy in classification decisions. Given a pre-trained ML model (e.g., an image classifier), conformal prediction augments its output: instead of one decision (e.g., the class of an image), it returns a *conformal set* (e.g., three possible classes), such that the correct answer is present in the conformal set with a target probability. Intuitively, better predictions are associated with smaller sets; on the other hand, accepting a larger conformal set may result in a higher confidence level.

The goal of conformal prediction is similar to ours, i.e., moving towards dependable ML; at the same time, the two approaches are orthogonal and, indeed, complementary. While conformal prediction seeks to *augment* the output of an already-trained model, our goal is to introduce reliability *within the learning process*, by achieving a target loss quantile. As also shown in our performance evaluation, the two methodologies can be profitably combined, as implementing the learning process with reliability in mind will also yield smaller conformal sets at the stage of inference.

Neural architecture search (NAS) [15] is a family of techniques aimed at automatically defining the best DNN architecture for a task at hand, subject to a variety of constraints. While the fundamental goal of NAS and DepL is the same, i.e., finding the model that best suits the task and the available resources, the two approaches are fundamentally different. Indeed, while NAS techniques design the DNN architecture from scratch and then train it from scratch, DepL is able to leverage a finite (but potentially very wide) set of (possibly) pre-trained models, refining them as needed.

## III. System Model and Problem Formulation

This section first introduces the model we develop to capture all relevant system's aspects (Sec. III-A) and then presents the mathematical formulation of the problem of dependable learning for DNNs (Sec. III-B).

### *A. 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 learning, (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 learning process. To encompass a wide range of distributed learning paradigms, including federated learning (FL) and sequential learning, we consider that the orchestrator identifies one or more clusters of learning nodes, with each cluster performing the vanilla FL [16], and, possibly, contributing to the learning 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-size or a compressed version). The scenario thus includes the following main elements:

- *Clusters of learning nodes*, $n \in \mathcal{N}$, that, without loss of generality, are assumed to include homogeneous nodes with equal communication and computational capability. Notice that there may be as many clusters as types of nodes with different capabilities – in the extreme case, even one cluster per node. We then 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 in the cluster. 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 an arbitrary 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 fine-tunes the model;
- *ML models*, possibly pre-trained, suitable for fine-tuning to fit the task at hand, in their full or compressed versions. They are represented as elements of the models set, $m \in \mathcal{M}$; importantly, elements in $\mathcal{M}$ can include variants of the same model (e.g., a full AlexNet and a pruned version thereof) as well as pairs of student-teacher models for which transfer learning techniques can be applied. 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 in epoch $k$ for fine-tuning model $m$ using data from dataset $d$.

Given a model $m \in \mathcal{M}$ and being cluster $n \in \mathcal{N}$ the one currently fine-tuning $m$ with dataset $d \subseteq \mathcal{D}$, let $l^{\text{run}}(k, d, m)$ be the global change in the loss achieved by fine-tuning model $m$ over $d$ in epoch $k$. Notice that the learning process aims at minimizing the loss, hence, $l^{\text{run}}$ will be negative if the learning 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 $l^{\text{sw}}$ 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-B, estimating them is an orthogonal problem to our own.

### *B. Decisions and problem formulation*

The orchestrator has to make the following decisions:

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

- Whether or not cluster $n \in \mathcal{N}$ should contribute to the learning at epoch $k$, expressed through binary variable $y(k,n) \in \{0,1\}$;
- Whether or not 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 fine-tuning model $m$ by a node of a selected cluster, expressed through the real variable $x(k,m,n) \leq y(k,n) r(n)$;
- 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.,

$$T(k) = \frac{c(k,d,m(k))}{x(k,m,n)} + 2\frac{s(m(k))}{b(n)}\,. \tag{1}$$

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

$$\begin{aligned} \ell(k,m) = \ell(k-1,m) + l^{\text{run}}(k,d,m(k)) \\ + l^{\text{sw}}(k,d,m(k-1),m(k)) \end{aligned} \tag{2}$$

where $\ell(0,m)$ is the initial loss value and $k$ is the current epoch. Note how the last term in the above equation reduces to zero if no model switching occurs from epoch $k-1$ to epoch $k$. Also notice how the initial loss $\ell(0,m)$ in (2) will be, in general, better for pre-trained models than for untrained ones.

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 $T^{\max}$:

$$\min_{x,y,z,K,m} \sum_{k=1}^{K} \sum_{n \in \mathcal{N}} \left( \sum_{d \in \mathcal{D}} \beta(d,n) z(d,n) + \kappa(n) x(k,m,n) \right) \tag{3}$$

$$\text{s.t.} \quad \ell_\omega(K,m) \leq \ell_\omega^{\max} \tag{4}$$

$$\sum_{k=1}^{K} T(k) \leq T^{\max} \tag{5}$$

$$x(k,m,n) \leq y(k,n) r(n) \quad \forall k,n \tag{6}$$

$$z(d,n) \leq y(k,n) \quad \forall d,n,k. \tag{7}$$

It is important to highlight that in (4) $\omega \in [0,1]$ indicates a specific *quantile* of the loss, as opposed to a loss expected value. By tweaking $\omega$, we are therefore able to enforce different levels of risk, depending upon the scenario and application at hand. Also, notice how (7) 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 learning, seeking to keep a target quantile $\omega$ 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 learning 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 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 [17] and feedback loops [18]. The impact of model selection on the overall learning is somehow better understood, though with significant uncertainty [5], and finding the right resources to run a given model can be mapped to many well-studied problems [19], [20]. 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 learning process. 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.

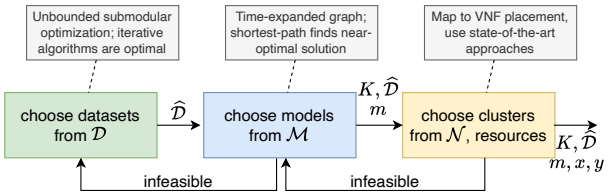


Fig. 3. The main steps of the DepL solution strategy. Grey callouts on top summarize our approach to tackle the individual steps.

### A. Dataset selection

Adding new datasets to improve learning quality may have undesirable consequences: (i) adding a new dataset implies that more fine-tuning 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 higher consumption of network resources. Accordingly, we begin with no selected datasets and add them as needed. It is worth noticing that, at this stage, we take the learning time as proxy of the overall learning 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 – under given resource assignment decisions, which are considered to be fixed at this stage – 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 learning 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 [21], [22]; (ii) The processing time, $\widehat{T}(\widehat{\mathcal{D}})$, grows 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:

$$\min_{\widehat{\mathcal{D}}\subseteq\mathcal{D}} \widehat{K}(\widehat{\mathcal{D}})\widehat{T}(\widehat{\mathcal{D}}), \tag{8}$$

which is an *unbounded submodular minimization problem* [23]. Such problems can be solved to optimality in polynomial time via several different algorithms. Examples range from the traditional ellipsoid method [24] (based on the notion of Lovász extension [25], linking submodular combinatorial problems to convex continuous ones) to more modern approaches [26], [27] that achieve *strongly polynomial* runtime, only depending upon the problem size.

In summary, regardless of the concrete algorithm used to solve (8), 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 the original problem in (3), which is NP-hard (as per Property 1). Specifically, the costs associated with the nodes using the dataset in $\widehat{\mathcal{D}}$ may be larger than the optimum. On the other hand, the number of such nodes will often be very small, and their cost will be outweighed by the cost of the nodes that only contribute to the learning process with their computational resources, i.e., such that $y(k,n) = 1$ even if $z(d,n) = 0$.

**Summary**. The dataset selection step of DepL (step 1 in Fig. 3) chooses which of the datasets in $\mathcal{D}$ to use for the learning process, i.e., sets the $z(d,n)$ variables. We remark that the problem to be solved at this step is an unbounded submodular minimization problem; hence, it can be solved to optimality through simple and efficient iterative approaches.

### 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 on the graph (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 $\eta$ 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-tuple $(m(k), k, T(k), \ell_\omega(k,m))$, where:

- $m(k)$ is a model in $\mathcal{M}$;
- $k$ is an epoch in $1,\dots,K$;
- $T(k){=}\frac{i}{\eta}T^{\max}$, with $i{=}1,\dots,\eta$, is the elapsed learning time in the state represented by the vertex;
- $\ell_\omega(k,m){=}\frac{j}{\eta}\ell(0,m)$, with $j{=}1,\dots,\eta$ and $\ell(0,m)$ being the initial loss value, reflects, in a similar way, the $\omega$-quantile of the current loss.

It is important to highlight the role of $\eta$ in choosing the most appropriate trade-off between solution accuracy and algorithmic complexity. In fact, as better detailed in Sec. V, higher values of $\eta$ result in solutions closer to the optimum, but also in larger graphs, hence, higher complexity.

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 learning time. This allows the size of the graph to be manageable, at the expense of precision. Concerning edges, we add an edge from vertex $(m(k), k, T(k), \ell_\omega(k,m))$ to vertex $(m(k{+}1), k{+}1, T(k{+}1), \ell_\omega(k+1,m))$ 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 $\ell_\omega(k+1,m)-\ell_\omega(k,m)$. 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 $\Omega$ 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 $\Omega$. Since vertices connected to $\Omega$ 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 $\Omega$ 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 $\eta$, we can reduce this issue and get arbitrarily close to the optimum, at the cost of 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 [28], [29], 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 $l^{\text{run}}$ 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 testing 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) can 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 – intuitively, because it is insufficient – 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.

**Summary**. The model selection step of DepL (step 2 in Fig. 3) chooses the model $m$ to use at each training epoch $k$, i.e., sets the $m(k)$ variables. To make such decisions, we (i) build a directed, weighted, time-expanded graph, where nodes correspond to potential decisions and the associated training times, and weights on the edges correspond to costs. The graph is built in such a way that only feasible solutions are represented therein, and performing a shortest-path search finds the optimal one.

### C. Node and resource allocation

Once the datasets and models are chosen, the only remaining decisions to make concern the clusters to use for learning and 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 [30] 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 fine-tuned 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.

**Summary**. The node and resource allocation step of DepL (step 3 in Fig. 3) chooses the node clusters $n\in\mathcal{N}$ to use for training, and allocates resources therein, i.e., sets the $x(k,m,n)$, $y(k,n)$, and $r(n)$ variables. DepL does not provide an ad-hoc strategy to perform this step, rather, we show how it can be reduced to a VNF placement problem, which has been widely studied in the literature and for which many efficient and effective solutions exist.

## 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), thus confirming the need for a heuristic solution;

- 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 (3) subject to constraints (4)–(7) is NP-hard.*

*Proof:* We prove NP-hardness by reduction from the knapsack problem, which is known [31] 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 polynomial 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 [27], 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 [30], which has a complexity of $O\left(\frac{1}{\varepsilon}\right)$, where $\varepsilon$ 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 $\eta$; specifically, as per [29, Theorem 4.3], the resulting competitive ratio is $\frac{1}{\eta}$. Finally, the approach in [30], 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.

**Dynamic scenarios.** Although, for simplicity, we state and prove DepL's complexity and performance guarantees in static scenarios, e.g., where the task to perform does not change over time, both are also valid in dynamic scenarios. Specifically:

- DepL's *constant* competitive ratio is not affected by the scenario size;
- DepL's *worst-case* computational complexity is polynomial in the largest size the scenario can take.

**DepL's impact on learning scalability.** Our analysis concerns the scalability and performance of the DepL solution concept, hence, it shows that DepL can make swift and high-quality decisions. This is a distinct concept from the scalability (indeed, the viability) of the learning process as a whole, e.g., whether it is possible to even learn from the available datasets in $\mathcal{D}$ using the models in $\mathcal{M}$. Such a scalability is influenced by many factors outside of DepL's control, e.g., the models' quality and the completeness of the datasets. At the same time, it is important to recall that, as per Property 3, DepL yields near-optimal decisions that make learning as cheap, swift, and effective as possible. In other words, if there is *any* combination of datasets, models, and nodes that result in a learning process consistent with the quality and speed requirements, DepL is guaranteed to find it.

## VI. Numerical Results

**Roadmap**. In this section, we validate both the performance of DepL and the observations and assumptions it is based upon. Specifically:

- Sec. VI-A describes our experimental settings and the benchmarks we consider;
- Sec. VI-B shows how real-world experiments can be used to estimate the $l^{\text{run}}$ and $l^{\text{sw}}$ parameters, which are an input to DepL;

- Sec. VI-C validates the diminishing-returns, submodular behavior of the relationship between the quantity of data and the learning performance, which we leverage for step 1 of DepL;
- Sec. VI-D compares the performance of DepL, the optimum, and a state-of-the-art approach in solving our full problem.

### A. Experimental settings

To assess the performance of our method, we perform image classification in a sequential learning scenario using two datasets, to wit, the classic CIFAR-10 benchmark and the real-world GTSRB road signal recognition dataset [32]. Specifically, we consider three nodes, each belonging to a different category reflecting their computational capabilities and located respectively in the cloud, edge, and far-edge.

The order in which the nodes contribute to the learning goes from the highest (cloud) to the lowest (far-edge) computing capability, as ultimately the model needs to be personalized with the data owned by IoT or user devices (which have indeed lower compute capabilities). At each learning stage, the model is further pruned, so as to make it amenable for learning at the lower-end nodes. More formally, the learning procedure starts at the cloud node, which fine-tunes the full model. Then, after $K_1$ epochs, the model is pruned with pruning ratio $F_1$ and sent to the edge node. The latter continues the learning and, after $K_2$ learning epochs, further prunes the model with ratio $F_2$ and hands it over to the remaining node.

This behavior, i.e., going from large to small models and from highest- to lowest-capability nodes, reproduces the most common – and, indeed, most sensible – way multi-model learning is performed. However, it is important to notice how the opposite can also happen, most notably, when moving from a pruned version of a model back to a full one. We do not consider such cases in our performance evaluation, owing to their comparative rarity; however, both our system model and the DepL solution concept are capable of supporting them.

In our reference scenario, each dataset $d \in \mathcal{D}$ is associated with one cluster $n \in \mathcal{N}$, i.e., the $\beta(d,n)$ parameter (representing the bandwidth available for data source-to-cluster transfers) is infinity with that cluster and zero with the others. This reproduces the very common case where data sources (e.g., sensors) can communicate directly only with nodes in a given cluster, and does not stem from a limitation of our system model or solution strategy. The nodes datasets include, respectively, 17,500, 12,500 and 7,500 samples, with the cloud and the edge ones having unbalanced datasets. The former has 2,750 for each of classes 1–6 and 250 for the remaining four classes; the latter node has 1,500 samples for each of classes 1–8 and 250 for classes 9 and 10; finally, the far-edge node has a balanced data distribution, with 750 samples per class. Also, we consider a second scenario in which the edge and far-edge nodes have swapped datasets. Notice that drawing all datasets in $\mathcal{D}$ from the same real-world dataset is simply a matter of expediency, as it makes our results simpler to obtain and easier to interpret, and does not come from a limitation of either our system model or solution strategy.

Two convolutional DNNs are considered, namely, AlexNet [9] and a variant of the ResNet18 [33] specifically tailored for the image dimension of the CIFAR dataset, i.e., $32\times32$. For the experiments, we employ all the 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 $F_1, F_2 \in \{0.25, 0.36, 0.5, 0.75\}$. As for datasets, both our system model and our solution strategy allow arbitrary elements in $\mathcal{M}$, not just pruned versions of the same DNN.

During the learning process, we set the batch size equal to 64; for the AlexNet model, we employ the gradient descent optimizer with learning rate and momentum equal to $10^{-3}$ and $0.9$, while for the ResNet18 we use the Adam optimizer with learning rate equal to $10^{-4}$.

Remarkably, we also perform conformal prediction, which creates a set of predicted classes such that the correct one lies in the set with a given probability. At each learning epoch, we compute the size of such sets for four values of probability, namely, 0.80, 0.90, 0,95, and 0.99.

### B. Estimating the loss evolution

As observed in Sec. II, the evolution of the testing loss over the learning epochs, $k$, cannot be predicted with certainty, rather, a probabilistic representation thereof should be used. Previous work [34] 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:

$$\ell^{\text{run}}(m,k) = \frac{A_m}{\sqrt{k+B_m}} \cdot x, \tag{9}$$

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 (9) 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.

Importantly, the data necessary to perform the parameter and distribution fitting are collected during the original training of the model, i.e., prior to its fine-tuning. We demonstrate how these tasks can be performed by considering two examples leveraging the CIFAR-10 dataset and the AlexNet and ResNet DNNs. Results with the GTSRB dataset, omitted for brevity, have shown exactly the same behavior, though – of course – with different coefficients.

Starting from AlexNet, Fig. 4(left) shows the empirical pdf of $x$ for an example testing loss curve, along with the fitted pdf $\mathcal{X}_m$, setting $K_1{=}17$, $K_2{=}60$, $F_1{=}0.5$, 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

[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.

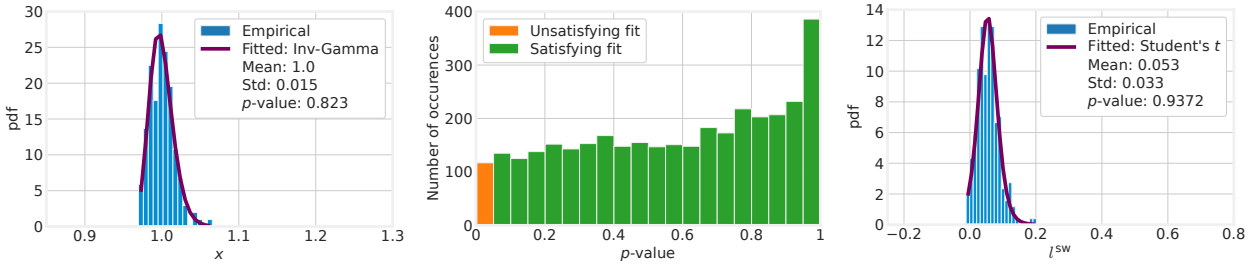


Fig. 4. AlexNet: Example distributions of $\mathcal{X}$ as defined in (9) along with the Inverse-Gamma fit (left); distribution of the $p$-values highlighting fitting quality (center); example distribution of $l^{\text{sw}}$ as defined in (2) along with the Student's t fit (right).

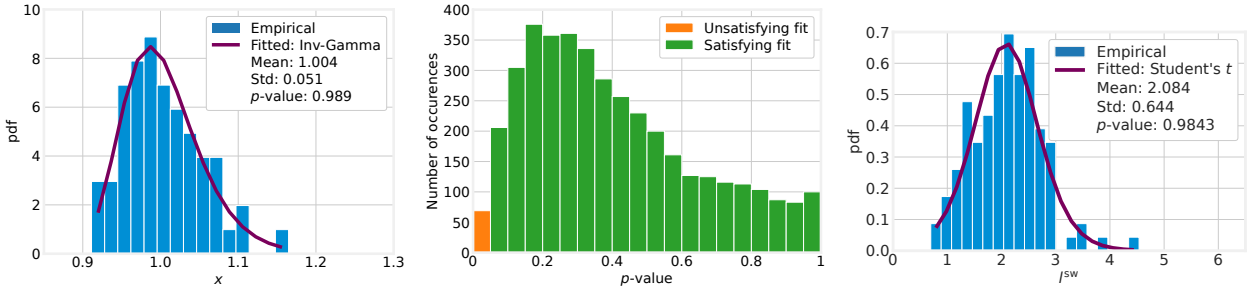


Fig. 5. ResNet DNN: Example distributions of $\mathcal{X}$ as defined in (9) along with the Inverse-Gamma fit (left); distribution of the $p$-values highlighting fitting quality (center); example distribution of $l^{\text{sw}}$ as defined in (2) along with the Student's t fit (right).

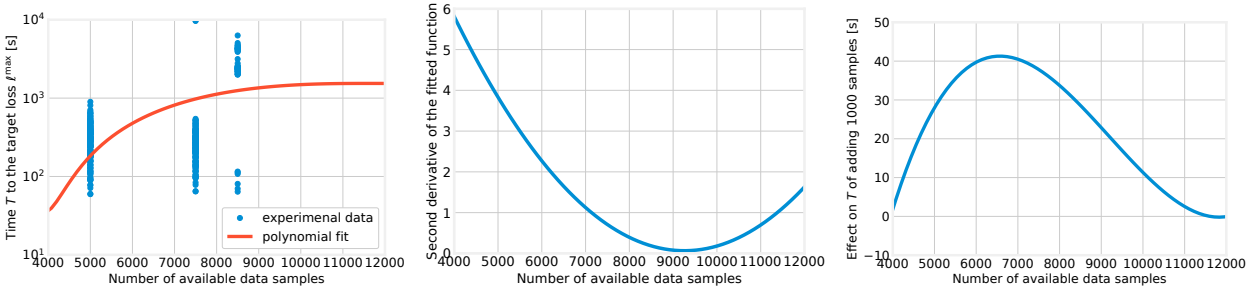


Fig. 6. Submodularity of $\widehat{K}(\widehat{\mathcal{D}})\widehat{T}(\widehat{\mathcal{D}})$: learning time as a function of the quantity of data used and polynomial best fit (left); second derivative of the fitted function (center); effect of adding 1,000 samples as a function of the already-available data (right).

Fig. 4(center) shows the $p$-values for the distributions obtained using all possible combinations of the number of epochs $K_1, K_2$, and pruning factors $F_1, F_2$ mentioned in Sec. VI-A: 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 $l^{\text{sw}}$, i.e., the difference between the testing 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 testing loss after model switching. Again, such an increase cannot be predicted 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.

Fig. 5 presents the same information, obtained using the ResNet DNN as a reference. Similar to Fig. 4, we can observe a very good fit between empirical data and fitted distributions, for the loss improvement (Fig. 5(left)) as well as the switch loss $l^{sw}$ (Fig. 5(right)). Fig. 5(center) confirms that the fit is very good in virtually all instances, with $p$-values abundantly exceeding the threshold of $0.05$.

### C. Effect of the quantity of data

Before we move to evaluating DepL's performance, we verify the property we exploit for the data selection phase of DepL (step 1 in Fig. 3), i.e., that the total learning time $\widehat{K}(\widehat{\mathcal{D}})\widehat{T}(\widehat{\mathcal{D}})$ is a submodular function of the size of chosen datasets $\widehat{\mathcal{D}}$. To this end, we perform a set of experiments using the ResNet DNN, equipping the nodes with different subsets of the CIFAR-10 dataset and observing the time it takes to reach a target loss value. It is worth mentioning that experiments performed with the AlexNet DNN and the GTSRB dataset have shown exactly the same behavior.

Fig. 6(left) shows the learning time as a function of the quantity of data, for different target loss values. In all cases we can observe the same behavior: the learning time first decreases (as more data results in a larger loss improvement per epoch) and then increases again (as more data results in longer epochs). Importantly, there is always *one* peak; adding more data once that peak is past never results in improved performance.

Fig. 6(center) illustrates the second derivative of the fitted functions: such derivative is always positive, hence, the functions themselves are convex – which is the condition we require for the first step of DepL to yield optimal decisions. Fig. 6(right) provides an alternative view of the same phenomenon, and portrays the gain of adding $1,000$ extra samples as a function of the dataset size. We can observe that such a gain steadily decreases, showing the "diminishing returns" behavior associated with submodular problems.

In summary, the behavior shown in Fig. 6 is consistent with the findings in the literature we leverage in the first step of DepL, and confirm that the decisions made therein are optimal.

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

### D. 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. Also, as better discussed in Sec. VII, we are not aware of any state-of-the art solutions that solve the same problem as DepL, most notably, that are able to jointly make decisions on the data to use, the models to use and when to switch between them, and the node and resources to exploit. Accordingly, we benchmark our DepL solution described in Sec. IV-B against the following 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 [5] (labelled as *best-exp* in plots).

The best-exp scheme makes decisions of the same nature as our DepL and in a way similar to DepL itself, i.e.,

1) identifying the possible model- and node-selection decisions;
2) summarizing their effects through an expanded graph;
3) selecting, through shortest-path search, the one resulting in the best performance.

Unless DepL, however, best-exp only 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 $\eta$=100 and $\varepsilon$=1.

For all strategies, we have three models in $\mathcal{M}$, namely, three versions of the AlexNet DNN or three versions of the ResNet DNN, whose features are summarized in Tab. I and Tab. I, respectively. Those models reproduce the most relevant aspect highlighted by our experiments in Sec. VI-B, i.e., the fact that the expected loss yielded by a certain model and the distribution of such loss are distinct and – to an extent – independent quantities. 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 testing loss performance; "slow-steady", on the other hand, provides slower but more reliable convergence, while "intermediate" yields balanced performance.

TABLE I
ALEXNET VARIANTS USED IN OUR EVALUATION, EXPECTED AND VARIANCE OF LOSS IMPROVEMENT PER EPOCH, AND RESOURCE DEMAND

| Model | Parameters | Exp. loss improvem. | Var. of loss improvem. | Norm. needed resources |
|---|---|---|---|---|
| Fast-unsteady | $A_m$=77, $B_m$=74 | 1.02 | 0.031 | 2.6 |
| Intermediate | $A_m$=17, $B_m$=3.6 | 1.00 | 0.010 | 1.4 |
| Slow-steady | $A_m$=7, $B_m$=0.3 | 0.98 | 0.007 | 1 |

TABLE II
RESNET VARIANTS USED IN OUR EVALUATION, EXPECTED AND VARIANCE OF LOSS IMPROVEMENT PER EPOCH, AND RESOURCE DEMAND

| Model | Parameters | Exp. loss improvem. | Var. of loss improvem. | Norm. needed resources |
|---|---|---|---|---|
| Fast-unsteady | $A_m$=57, $B_m$=81 | 0.99 | 0.021 | 2.3 |
| Intermediate | $A_m$=10, $B_m$=4.2 | 0.98 | 0.008 | 1.1 |
| Slow-steady | $A_m$=6, $B_m$=0.7 | 0.97 | 0.006 | 0.9 |

We consider a network infrastructure including two types of servers (hence, clusters nodes in $\mathcal{N}$): A-class nodes, with capability 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 [30].

Starting from the AlexNet DNN, 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 (3). The results are summarized in Fig. 7(left), showing the cost incurred for different target values of the loss 99th percentile. As one might expect, tighter loss requirements require more learning, 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. 7(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 (4) with an equality sign, i.e., they provide exactly the required value of the loss 99th percentile. On the other hand, Fig. 7(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

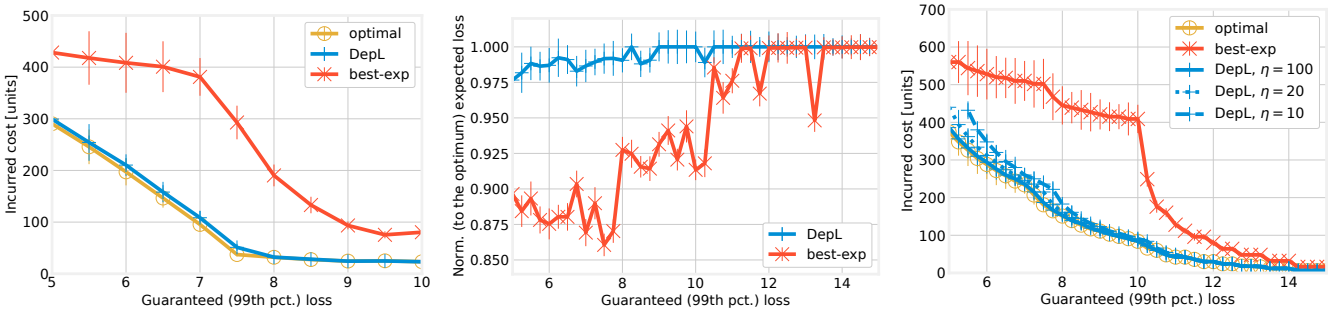


Fig. 7. AlexNet: Performance of DepL and its benchmarks. Cost (left); normalized expected loss (center); effect of $\eta$ (right).

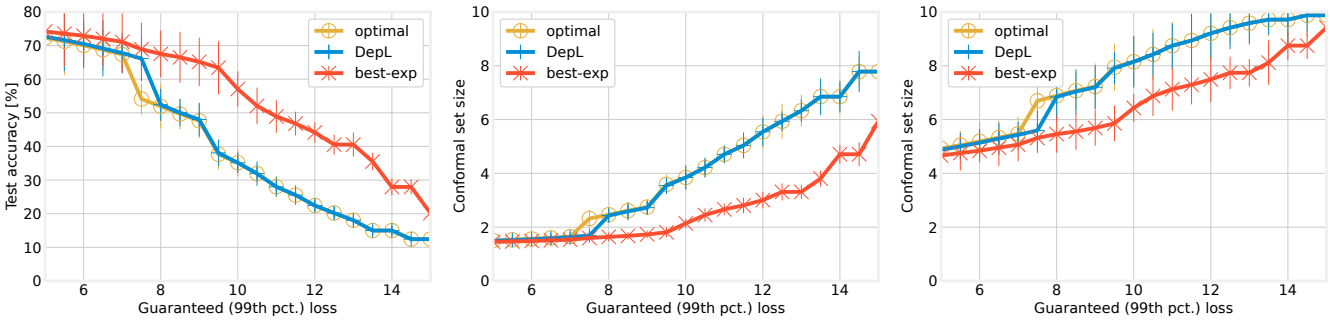


Fig. 8. AlexNet: Performance of DepL and its benchmarks. Accuracy (left); size of the conformal set at 80% (center) and 99% (right) confidence level.

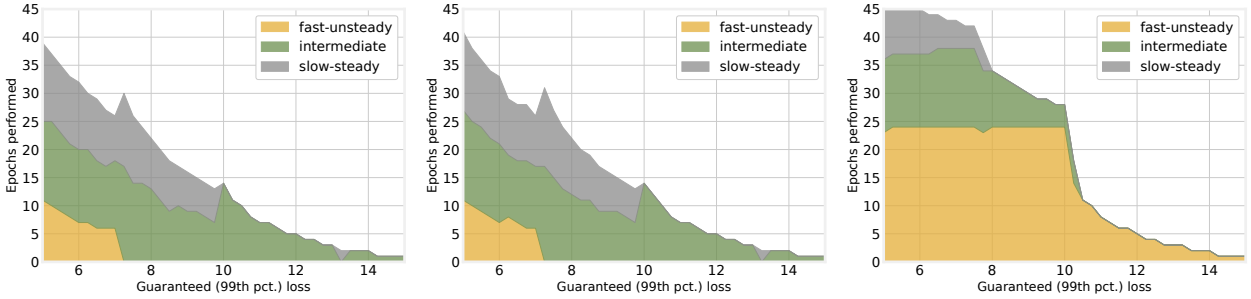


Fig. 9. AlexNet: Epochs spent with each model by the optimal solution (left), DepL (center), and *best-exp* (right).

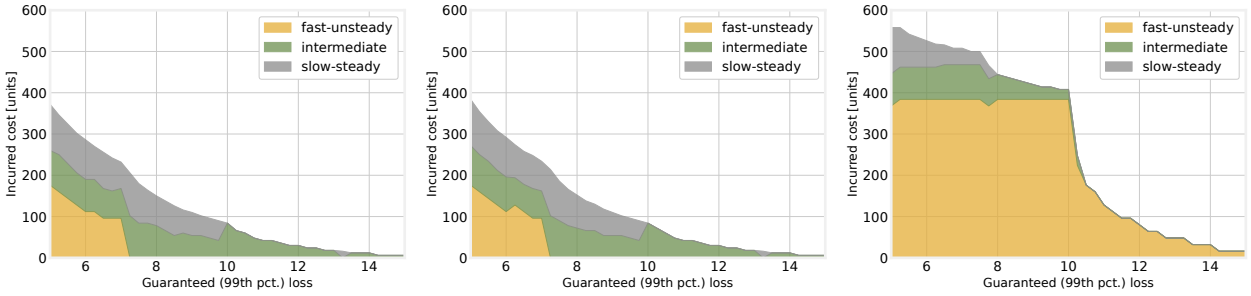


Fig. 10. AlexNet: Cost incurred with each model by the optimal solution (left), DepL (center), and *best-exp* (right).

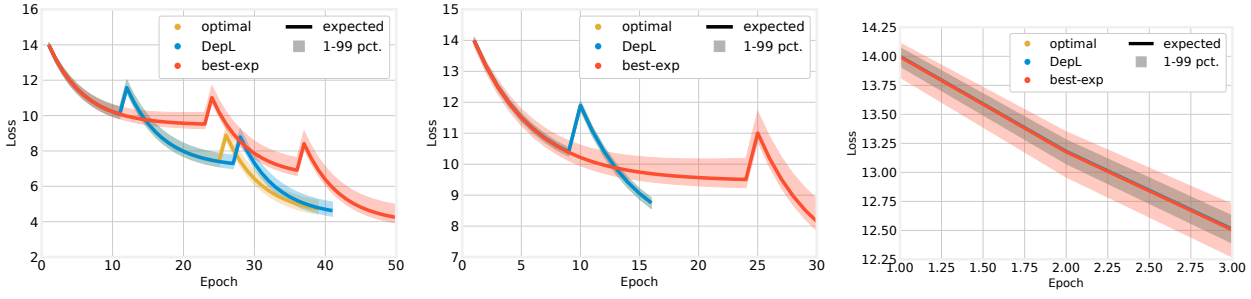


Fig. 11. AlexNet: Evolution of the expected and guaranteed loss when the guaranteed loss is 5 (left), 9 (center), and 13 (right).

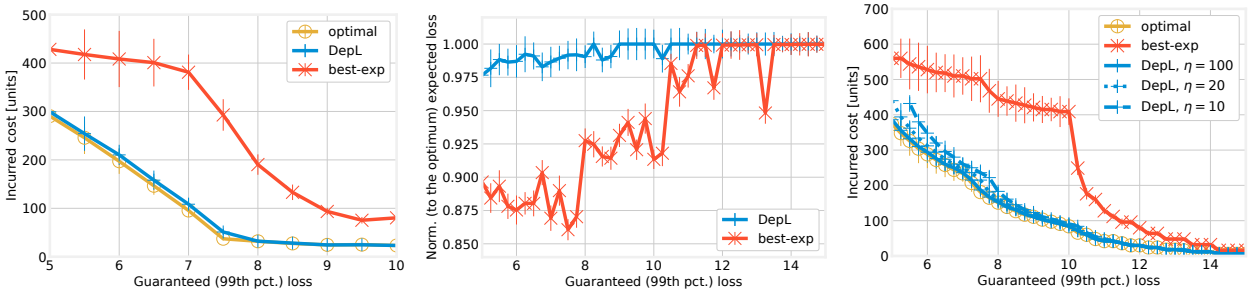


Fig. 12. ResNet, GTSRB dataset: Performance of DepL and its benchmarks. Cost (left); normalized expected loss (center); effect of $\eta$ (right).

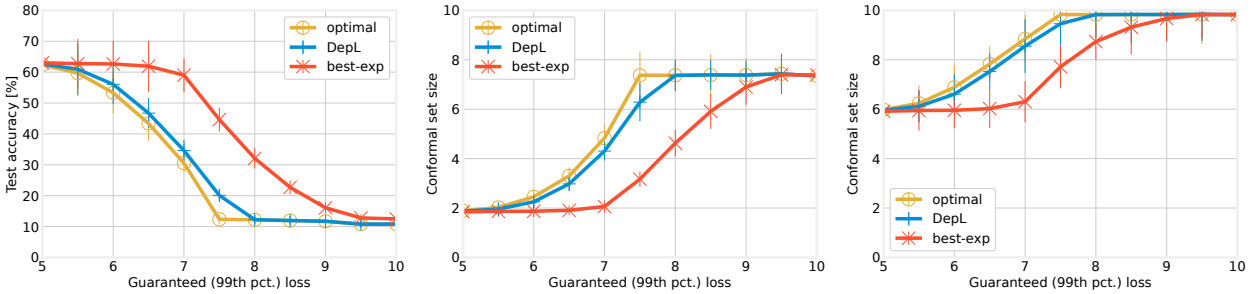


Fig. 13. ResNet, GTSRB dataset: Performance of DepL and its benchmarks. Accuracy (left); size of the conformal set at 80% (center) and 99% (right) confidence level.

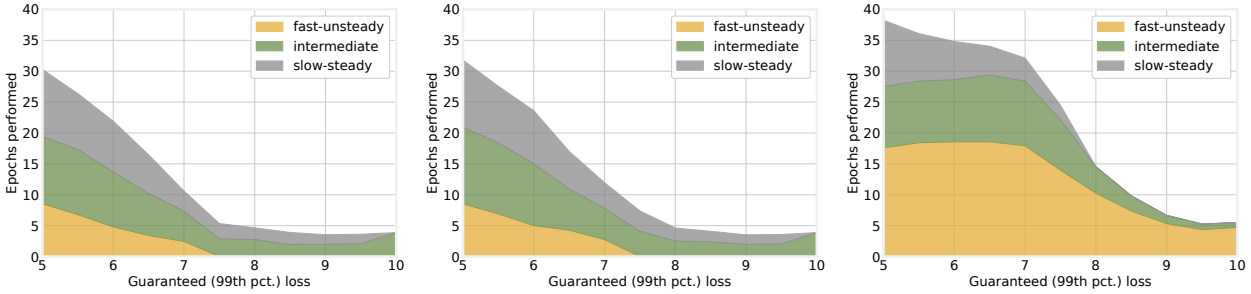


Fig. 14. ResNet, GTSRB dataset: Epochs spent with each model by the optimal solution (left), DepL (center), and *best-exp* (right).

expected value brings no benefit. Indeed, the reason why DepL almost matches the optimum is its ability to focus on the target quantile of the loss, disregarding its expected value.

DepL has two main parameters, to wit, $\eta$ and $\varepsilon$, impacting both its optimality and computational complexity. Among the two, $\eta$ has the largest impact, since[3] it appears raised to a higher exponent in Property 2. Accordingly, in Fig. 7(right), we assess the effect of decreasing the parameter $\eta$, which we use when building the expanded graph in Sec. IV-B, on the performance of DepL. We observe that even a tenfold decrease in $\eta$ has just a modest impact on the DepL performance, which remains consistently close to the optimum and much better than best-exp. This highlights 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.

We now move to Fig. 8, giving a more concrete description of the performance achieved by DepL and its counterparts. Fig. 8(left) shows the classification accuracy of the model for different levels of guaranteed loss. The first observation we can make is that, as one might expect, higher loss corresponds to lower accuracy. More interestingly, and consistently with Fig. 7(center), Fig. 8(left) shows that DepL yields a *lower* accuracy than best-exp. Indeed, as long as the necessary performance (in our case, the loss percentile) is achieved, there is no need to continue the learning further, and doing so would result in a waste of resources. The efficiency of DepL can be further observed by comparing Fig. 8(left) to Fig. 7(left) just above: in the most challenging conditions (i.e., when the guaranteed loss is lowest), DepL achieves essentially the same performance of best-exp, at a cost that is almost 40% lower.

In Fig. 8(center) and Fig. 8(right) we move to the realm of conformal prediction, and depict the size of the conformal set when the required confidence level is (respectively) 80% and 99%. Recall that smaller conformal sets correspond to better performance; taking as a reference the 80% confidence level (i.e., Fig. 8(center)), its size goes from 2 to 8 elements, as the loss requirement becomes less stringent. Consistently with Fig. 8(left), DepL – and, importantly, the optimum – result in larger conformal sets than best-exp, as the latter fine-tunes the DNN beyond the required quality. It is even more interesting to observe that, moving from the accuracy (Fig. 8(left)) to conformal sets for with medium confidence

[3]Experimental results (omitted for brevity) have confimed this observation.

level (Fig. 8(center)) and then to conformal sets with a high confidence level (Fig. 8(right)), the performance of DepL and its counterparts get *closer* to each other. This suggests that, when conformal predictions and/or high confidence levels are required, we are able to reap the full advantages of DepL efficiency (Fig. 7(left)), with only a minor difference in performance.

In Fig. 9, 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 fine-tuned. DepL (center plot) makes very similar choices to the optimum (left plot), consistently with Fig. 7; 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. 7(left).

Fig. 10 shows the total cost of each strategy, broken down according to the model they use. As in Fig. 7(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. 9) *and* spending more in each of them (Fig. 10), while obtaining a needlessly low expected loss (Fig. 7(center)).

Finally, Fig. 11 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. 9 and Fig. 10: 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. 11 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.

Next, we move to the ResNet DNN. As mentioned, we consider the three pruned versions of the model with factors (respectively) $F_1$=0.5, $F_2$=0.75, $F_3$=0.9 (see Tab. II). Furthermore, we now use the GTSRB dataset instead [32], including about 50,000 real-world pictures of road signs in Germany. The task to perform is still image classification (with the purpose of recognizing the signs), however, GTSRB includes 43 classes compared to CIFAR-10's ten. Our high-level purpose is to ascertain whether DepL performs in a (qualitatively) similar manner under different DNN architectures and datasets.

Fig. 12(left), summarizing the resource consumption of DepL and its counterparts, paints a similar picture to Fig. 7(left). DepL is vastly more efficient than best-exp and virtually matches the optimum. The difference is more clear in more challenging scenarios (when the target loss is smaller), and decreases as requirements become less stringent. We can also observe that, in general, achieving the same level of guaranteed loss is cheaper (i.e., takes less resources) with ResNet than with AlexNet (Fig. 7(left)). This is indeed consistent with the design goals of ResNet, which emphasizes efficiency and has a smaller number of parameters compared to AlexNet.

We can further observe, from Fig. 12(center), that DepL's better efficiency is achieved by avoiding unnecessary learning, resulting in a higher loss than best-exp (though still lower than the optimum). Comparing Fig. 12(center) to Fig. 7(center), we can also remark that the difference between the strategies is less dramatic (lines are closer to 1). This suggests that, due to its focus on efficiency, ResNet is more resilient than AlexNet to pruning. Also, DepL's efficiency advantage over best-exp significantly exceeds the latter's performance advantage over DepL (see Fig. 12(center) vs. Fig. 12(left)). Finally, Fig. 12(right) underlines that the effect of $\eta$ over DepL's performance is fairly limited, except for very tight performance requirements. This underscores that further room to improve DepL's efficiency exists if efficiency-oriented architectures like ResNet are used.

As for accuracy, shown in Fig. 13(left), it tends to be lower than the one achieved with AlexNet (Fig. 8(left)), even for the same loss values, and tends to decline faster as the guaranteed loss increases. This highlights that particular care must be put in selecting the loss targets; however, those are merely input data for all schemes. The relative performance of the strategies is consistent with both Fig. 8(left) and Fig. 12(left), i.e., DepL – like the optimum – achieves lower cost by avoiding unnecessarily fine-tuning the DNN.

Similar observations can be made about the size of conformal sets, depicted in Fig. 13(center) and Fig. 13(right) for (respectively) 80% and 99% confidence levels. When loss requirements are either very tight or very loose, all schemes achieve similar performance and make similar decisions; intermediate loss targets result instead in more varied behavior. In these cases, best-exp tends to achieve better learning performance (hence, smaller conformal sets); however, such extra performance is beyond the requirements and brings no additional advantage, while costing extra resources, as shown in Fig. 12(left). By comparing the $y$-axes of Fig. 13 and Fig. 8, we can notice that the size of conformal sets is larger for the GTSRB dataset than for the CIFAR-10 one, in spite of the accuracy being better. However, we need to recall that GTSRB has many more classes than CIFAR-10 (43 vs. 10): compared to the number of existing classes, the conformal set sizes for GTSRB are actually smaller than for CIFAR-10.

Finally, Fig. 14 summarizes how different strategies combine the model variants reported in Tab. II. Similar to Fig. 9, we can observe how best-exp (right plot) relies on the fast-unsteady model to a greater extent than DepL and the optimum. Interestingly, a large difference in model selection decisions (Fig. 14) results in a relatively small difference in learning performance (Fig. 13): this is due to the fact that,

as also presented in Tab. II, different ResNet variants tend to yield relatively similar loss improvements.

## VII. Related work

DepL is related to three main research areas: resource-aware distributed DNN training and fine-tuning, DNN model compression, and robust learning for DNN.

Distributed DNN training [35], i.e., using the resources and data of multiple nodes to perform a single learning task, is one of the foremost issues in ML research. Besides simpler approaches like federated learning [18], [36], recent works have advocated block-wise learning [37], where traditional backpropagation steps are dispensed and different blocks can exchange information [38]. In the former case, the goal is to locate and exploit the best resources for a given learning task. The main decision is often choosing the nodes to involve in the learning process, based on their speed [2], [3], the quantity [2], [36], [39] and quality [4], [36] of their data, the speed and reliability of their network [3], [40], as well as trust [41]. In all cases, the ML model is assumed to be given and fixed. Recent works [1], [42] target more complex scenarios, where layers of the DNN can be duplicated at different learning nodes if needed. The studies in [5], [43] also seek to adapt the learning task to the available resources, choosing from different existing models. In this context, significant effort has been devoted to predicting, via mathematical modeling, the performance and convergence of a DNN training task. Approaches include deriving convergence bounds [34], leveraging information on the cooperation of learning nodes [44], and accounting for information processing constraints [45].

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 [46], [47], by observing the evolution of the parameter values. Other compression techniques include knowledge distillation [48], 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 [49] and domain adaptation [50].

Robustness in distributed learning for DNNs is an emerging topic. Most works set in a FL scenario and focus on node selection; intuitively, the goal is to find the learning nodes that are most likely to provide reliable, high-quality model updates [51]. Such a robustness objective is balanced against fairness consideration in [52], aiming at distributing the learning load as evenly as possible. Robustness considerations are also combined with communication efficiency in [53], [54] and data quality in [51], [54]. Finally, [55] aims to improve robustness by identifying (and excluding) malicious nodes, which intentionally provide wrong updates, or no update at all. Importantly, all the aforementioned works deal with node selection; the reliability of the DNN model and the learning process are never accounted for.

Conformal prediction as a methodology to augment the output of already-trained DNNs has been introduced in the seminal paper [14]. Subsequent works have sought to address the main limitations of that work, to wit, its focus on classification tasks and a restricted set of supported loss functions. As an example, [56] extends the scope of conformal classification to arbitrary monotone loss functions, and [57] explores how conformal prediction can be used in regression tasks. Other recent works like [58] explore how to use conformal prediction to fix shortcomings in the input data, e.g., noisy labels.

In summary, DepL differs from the above works in three main ways. First, it performs *mutual* adaptation between available data sources, models to use, and resources to exploit. Second, it is able to use *multiple* models (or versions thereof) to achieve a given learning objective, exploiting model compression. Third and most important, and unlike *all* the works we have surveyed, DepL achieves *dependable* learning for DNN, i.e., it guarantees that a target learning quality is reached with a target probability.

Finally, a preliminary conference version of this work has appeared in [59]. Compared to [59], this paper has a wider scope (including conformal prediction), additional analysis on the submodular relationship between data quantity and prediction quality, and a much more thorough performance evaluation.

## VIII. Conclusions

We have sought to attain dependability in the distributed learning for DNN models. To achieve this goal, we have proposed a solution strategy called DepL, making decisions on the data, nodes, and models to use – as well as the times at which to switch from a model to another. By accounting for the mutual interference of the aforementioned decisions, DepL is able to make near-optimal decisions, whilst keeping a low (namely, polynomial) computational complexity. Our performance evaluation, based upon state-of-the-art, *de facto* standard dataset and DNN models, further confirms that DepL performs very close to the optimum. Results, obtained for both the CIFAR-10 and the GTSRB dataset, show that, with respect to current approaches, DepL can reduce the learning cost by 40–80%, depending on the desired loss guarantees, and the number of required learning epochs by 30%.

## References


[1] 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.

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

[3] 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.

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

[5] 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.

[6] A. Achille, A. Golatkar, A. Ravichandran, M. Polito, and S. Soatto, "Lqf: Linear quadratic fine-tuning," in *IEEE/CVF CVPR*, 2021.

[7] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," *arXiv preprint arXiv:2403.14608*, 2024.

[8] H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and T. Zhao, "Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization," *arXiv preprint arXiv:1911.03437*, 2019.

[9] 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.

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

[11] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[12] 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.

[13] 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.

[14] A. N. Angelopoulos, S. Bates, J. Malik, and M. I. Jordan, "Uncertainty sets for image classifiers using conformal prediction," *ICLR*, 2021.

[15] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, 2019.

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

[17] 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.

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

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

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

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

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

[23] S. Iwata, "Submodular function minimization," *Mathematical Programming*, 2008.

[24] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, 1981.

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

[26] J. B. Orlin, "A faster strongly polynomial time algorithm for submodular function minimization," *Mathematical Programming*, 2009.

[27] T. Itoko and S. Iwata, "Computational geometric approach to submodular function minimization for multiclass queueing systems," in *International Conference on Integer Programming and Combinatorial Optimization*, 2007.

[28] J. Martín-Pérez, F. Malandrino, C. F. Chiasserini, M. Groshev, and C. J. Bernardos, "KPI guarantees in network slicing," *IEEE/ACM Transactions on Networking*, 2021.

[29] G. Xue, A. Sen, W. Zhang, J. Tang, and K. Thulasiraman, "Finding a path subject to many additive qos constraints," *IEEE/ACM Transactions on networking*, 2007.

[30] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the nfv service distribution problem," in *IEEE INFOCOM*, 2017.

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

[32] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The german traffic sign recognition benchmark: a multi-class classification competition," in *IEEE JCNN*, 2011.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.

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

[35] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM CSUR*, 2020.

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

[37] A. Cheng, H. Ping, Z. Wang, X. Xiao, C. Yin, S. Nazarian, M. Cheng, and P. Bogdan, "Unlocking deep learning: A bp-free approach for parallel block-wise training of neural networks," in *IEEE ICASSP*, 2024.

[38] C. Yin, M. Cheng, X. Xiao, X. Chen, S. Nazarian, A. Irimia, and P. Bogdan, "Leader-follower neural networks with local error signals inspired by complex collectives," *arXiv preprint arXiv:2310.07885*, 2023.

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

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

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

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

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

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

[45] M. R. Znaidi, G. Gupta, and P. Bogdan, "Secure distributed/federated learning: prediction-privacy trade-off for multi-agent system," in *IEEE ISIT*, 2022.

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

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

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

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

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

[51] A. Ghosh, J. Hong, D. Yin, and K. Ramchandran, "Robust federated learning in a heterogeneous environment," *arXiv preprint arXiv:1906.06629*, 2019.

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

[53] 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.

[54] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, 2019.

[55] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen, "Learning to detect malicious clients for robust federated learning," *arXiv preprint arXiv:2002.00211*, 2020.

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

[57] R. Lunde, E. Levina, and J. Zhu, "Conformal prediction for network-assisted regression," *arXiv preprint arXiv:2302.10095*, 2023.

[58] M. Sesia, Y. Wang, and X. Tong, "Adaptive conformal classification with noisy labels," *arXiv preprint arXiv:2309.05092*, 2023.

[59] F. Malandrino, G. Di Giacomo, M. Levorato, and C. F. Chiasserini, "Dependable distributed training of compressed machine learning models," in *IEEE WoWMoM*, 2024.