Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation

Doctoral Program in Electrical, Electronics and Communications Engineering
(37^{th}cycle)

Resource-Aware and Resilient Learning in Mobile Networks

By

Giuseppe Di Giacomo

******

Supervisor:
Prof. Carla Fabiana Chiasserini

Doctoral Examination Committee:
Dr. Chiara Boldrini, CNR
Prof. Roberto Garello, Politecnico di Torino
Prof. Jaime Llorca, Università degli Studi di Trento
Dr. Alessandro Nordio, CNR
Prof. Elad Michael Schiller, Chalmers University of Technology

Politecnico di Torino
2025

Acknowledgements

I would like to thank everyone I met along this long and enriching journey.

First, I would like to express my gratitude to my advisor Prof. Carla Fabiana Chiasserini for her invaluable and continuous support, which helped me grow as a researcher. I am also thankful to Dr. Francesco Malandrino for his guidance and assistance throughout these three years.

I would like to thank EURECOM for hosting me as a visiting Ph.D. student; most of all, I am grateful to Prof. Pietro Michiardi and Dr. Giulio Franzese for their precious support and mentorship. In addition, I want to extend my gratitude to Prof. Marco Levorato from the University of California, Irvine, for his contribution and insightful comments during our collaboration.

I also want to thank the DET Department, the Qatar National Research Fund and the projects RAINBOW, CONNECT, CENTRIC, RESTART, PREDICT-6G and ADROIT6G, which supported my work.

To my lifelong friends, Gabri, Des, Luca, Nina and Federica, thanks for sharing difficult moments, and, above all, especially the countless happy ones. Your friendship means a lot to me.

To my friends and colleagues met during the Ph.D., Federico, Greta, Antonio, Franco, Riccardo, Tuna, Barbara, Saverio, Andrea, Lorenzo and Leonardo, thanks for all the lunches and coffee breaks, making the office feel a little more like home.

Thanks to my long-distance friends, Viola, Francesco and Piera; when we meet, it always feels like no time has passed since our days together in Rue Bricka.

A heartfelt thanks to Stella for being always there during this final year.

Last but not least, I am deeply grateful to my family and especially my parents, who have supported me all these years and allowed me to reach this great milestone.

Publications

Journal publications

- Dang, V. N., et al. (2022). Vessel-CAPTCHA: an efficient learning framework for vessel annotation and segmentation. *Medical Image Analysis*, 75, 102263.
- Malandrino, F., Chiasserini, C. F., **Di Giacomo, G.** (2022). Efficient distributed DNNs in the mobile-edge-cloud continuum. *IEEE/ACM Transactions on Networking*, 31(4), 1702-1716
- Malandrino, F., **Di Giacomo, G.**, Karamzade, A., Levorato, M., Chiasserini, C. F. (2023). Tuning DNN Model Compression to Resource and Data Availability in Cooperative Training. *IEEE/ACM Transactions on Networking*, 32(2), 1600-1615
- Malandrino, F., **Di Giacomo, G.**, Karamzade, A., Levorato, M., Chiasserini, C. F.. Achieving Machine Learning Dependability Through Model Switching and Compression. Submitted to: *IEEE Transactions on Mobile Computing.*

Conference publications

- **Di Giacomo, G.**, Härri, J., Chiasserini, C. F. (2022, October). Edge-assisted gossiping learning: Leveraging v2v communications between connected vehicles. In 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC) (pp. 3920-3927). IEEE
- Malandrino, F., Chiasserini, C. F., **Di Giacomo, G.** (2022, March). Energy-efficient Training of Distributed DNNs in the Mobile-edge-cloud Continuum. In 2022 17th Wireless On-Demand Network Systems and Services Conference (WONS) (pp. 1-4). IEEE

- Malandrino, F., **Di Giacomo, G.**, Karamzade, A., Levorato, M., Chiasserini, C. F. (2023, May). Matching DNN compression and cooperative training with resources and data availability. In IEEE INFOCOM 2023-IEEE Conference on Computer Communications (pp. 1-10). IEEE

- Morra, L., et al. (2023, May). Designing Logic Tensor Networks for Visual Sudoku Puzzle Classification. In NeSy (pp. 223-232).

- **Di Giacomo, G.**, Franzese, G., Cerquitelli, T., Chiasserini, C. F., Michiardi, P. (2023, December). Multi-View Latent Diffusion. In 2023 IEEE International Conference on Big Data (BigData) (pp. 6152-6154). IEEE

- Malandrino, F., **Di Giacomo, G.**, Karamzade, A., Levorato, M., Chiasserini, C. F.. Dependable Distributed Training of Compressed Machine Learning Models. In 2024 IEEE 25th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Perth, Australia, 2024 pp. 147-156

- **Di Giacomo, G.**, Malandrino, F., Chiasserini, C. F. (2024, June). Generosity Pays Off: A Game-Theoretic Study of Cooperation in Decentralized Learning. In 2024 IEEE International Conference on Communications Workshops (ICC Workshops) (pp. 105-110). IEEE.

- **Di Giacomo, G.**, Franzese, G., Cerquitelli, T., Chiasserini, C. F., Michiardi, P. (2024). DiMViS: Diffusion-based multi-view synthesis. In ICML 2024 Workshop on Structured Probabilistic Inference & Generative Modeling

- **Di Giacomo, G.**, Franzese, G., Cerquitelli, T., Chiasserini, C. F., Michiardi, P. (2024). DiMViDA: Diffusion-based Multi-View Data Augmentation. In IEEE CAMAD 2024. IEEE

Abstract

In the last decade, Machine Learning (ML) and, more specifically, Deep Learning have gained significant prominence, driving innovation across a variety of domains. ML models are commonly trained in a centralized manner, requiring extensive computational resources and large volumes of data to be aggregated on a central server, which also raises privacy concerns. To tackle these issues, the Cooperative Learning (CL) paradigm has emerged as a promising solution: it leverages the computational capabilities of a plethora of heterogeneous nodes, enabling local training while exchanging the models' parameters, without sharing sensitive raw data with third parties.

Many factors characterize the cooperative training, such as the choice of the model, the data to use and the nodes to employ; making decisions about these aspects is challenging, as they affect one another. The first part of this work presents two distinct CL scenarios. In the first one, the layers of a Deep Neural Network (DNN), or multiple instances of them, are run at different devices of the mobile-edge-cloud continuum. The second scenario focuses on sequential learning across sets of nodes and capitalizes on pruning, a well-established technique to compress DNNs, which requires additional decisions about when and how much to prune the model during the training. Importantly, each node trains all the layers of the model, whether full or pruned. For each scenario, we design an algorithmic framework that, given the interdependencies among the abovementioned factors, makes joint decisions about them to optimize the training energy consumption while meeting time and quality constraints. The proposed frameworks have polynomial time complexity and are proven to make near-optimal decisions, outperforming alternative methods, as validated through our extensive performance evaluation. Another key challenge in CL lies in the lack of incentive for nodes to participate in the learning, as they will not allocate their computational and communication resources for training unless it is beneficial to them. Thus, to foster cooperation among nodes, we develop a

game-theoretic approach based on the Generous Tit-for-Tat strategy. The designed method, which accounts for the heterogeneity of both nodes' resources and models to be trained, constitutes a Nash equilibrium and converges to the Pareto optimal operating point.

In the second part of this thesis, we focus on Generative Artificial Intelligence models owing to their ability to generate high-quality synthetic samples, which have been demonstrated to improve the performance of DNNs. In line with this, we first design a diffusion model for Novel Views Synthesis, endowed with the capability of accommodating a flexible number of input views of an object to generate multiple novel views thereof. The model outperforms the alternative state-of-the-art methods by up to 15% in terms of perceptual similarity between the generated and ground truth views. Finally, we employ a smaller-scale model with a similar architecture to generate synthetic novel views that are used to augment the training set in the context of image classification, resulting in higher accuracy. Additionally, we propose a method to combine synthetic and real views also at inference time, achieving further improvements. Overall, augmenting both the training and inference pipelines leads to an accuracy gain of up to 20%. Importantly, our inference scheme can also be applied to enhance the resilience in edge-assisted classification systems where the inference task is offloaded to an edge node.

Contents

Chapter 1

Introduction

In 2024, the Nobel Prize in Physics was awarded to John J. Hopfield and Geoffrey E. Hinton for their "foundational discoveries and inventions enabling machine learning with artificial neural networks" [1]. Although this event stirred some debate in the academic community, this award is solid proof of the impact and importance of Machine Learning (ML) – and in particular Deep Learning, a sub-branch of ML – in the current society. The past decade has witnessed the rapid development of Machine Learning, which has been driving significant transformations in many fields, such as healthcare, autonomous systems, conversational agents and finance, and is expected to become even more ubiquitous and relevant in both industry and academia. Though Hopfield and Hinton's research has been conducted starting from the 1980s, the rise of ML is more recent and can be attributed to two key factors: the development of increasingly powerful hardware, particularly the GPUs (Graphics Processing Units), and the availability of large amounts of data.

GPUs, as opposed to CPUs, are optimized for parallel computing, typical in ML and Deep Learning, resulting in significant reductions in both training and inference times. As a consequence, GPUs have allowed researchers and practitioners to design increasingly deeper and more complex models, while processing faster large quantities of data. In fact, the availability of large-scale datasets is considered another fundamental aspect driving the growth of ML, leading to powerful models that exhibit growing generalization capability.

In 2012, the AlexNet Convolutional Neural Network (CNN) [2] won the ImageNet Large Scale Visual Recognition Challenge [3], a prestigious past competition focused

on computer vision tasks, namely image classification and object detection. AlexNet marked a significant breakthrough in Machine Learning, particularly in computer vision. Notably, its state-of-the-art performance was the result of a deep architecture, which was made possible through the use of GPUs, the training on the large-scale ImageNet [4] dataset and the use of other techniques, such as the ReLU activation function.

Datasets like ImageNet are specifically built for the training of Deep Neural Networks (DNNs), aiming to further increase their performance and generalization capabilities; still, we could benefit from the great variety of data that is automatically generated by a plethora of devices capable of rapidly collecting extensive volumes of information. Valuable data sources include smartphones, wearable devices, IoT sensors, cameras, vehicles, medical records, and many more. The large scale and diversity of such data present a huge potential to expand the application of ML across various domains.

However, traditional centralized training requires such large volumes of data to be aggregated and processed on a central server, which necessitates significant computational power; furthermore, data may contain sensitive information, raising concerns about privacy. To address these issues, the Cooperative Learning (CL) paradigm has emerged as a promising approach. CL leverages the computational power of multiple nodes that collaboratively train or fine-tune a model by using their local private data. Such nodes exchange the updated model weights or their gradients, instead of transmitting their raw data, thereby mitigating privacy issues. As an alternative, each node could train its own model independently. However, this strategy may suffer from suboptimal learning outcomes due to limited local datasets [5]. Cooperative learning, instead, could achieve better performance, by leveraging a larger total amount of data without requiring it to be shared. Notably, the model parameters and gradients could still leak some private information; for this reason, practical implementations integrate techniques specifically developed to reduce this risk, such as differential privacy [6].

Cooperative Learning is a broad term that includes a wide range of approaches. Among these, the *de facto* standard algorithm is Federated Learning [7], where nodes are assisted by an aggregator, typically referred to as parameter server. Iteratively, the aggregator transmits the latest global model to the learning nodes, which perform the local training and send back the gradients or the updated model weights. The

aggregator then combines the received parameters to obtain the new global model and starts again the procedure. Federated Learning falls under the Distributed Learning paradigm, a term that refers to those techniques where learning nodes are assisted by a central coordinator, which may either aggregate the models' parameters or make key decisions about the learning process. The other main popular CL scheme is referred to as Decentralized Learning, which eliminates the need for a coordinator, with nodes directly exchanging the parameters with one another. Although in literature the terms "distributed learning" and "decentralized learning" are often used interchangeably, in this thesis we will adopt this terminology, which is aligned with other works, such as [8].

Nevertheless, CL is characterized by many research questions that must be addressed. First, CL inherits, and can even exacerbate, some of the challenges of centralizing training, such as generalization capability and training costs. For instance, Google plans to buy nuclear power to cope with the exploding amount of energy required to power the company's Artificial Intelligence (AI) data centers [9]. Therefore, reducing the energy consumption entailed by the training of complex ML models is a crucial research topic, due to its great impact on both economic cost and environmental sustainability. Such a challenge has driven the first part of our research; specifically, we will analyze the main factors characterizing two different CL scenarios and design resource-aware algorithmic frameworks that make decisions about the cooperative training aimed to minimize energy costs, subject to quality and time constraints.

Second, CL introduces new drawbacks, such as systems and statistical heterogeneity, network latency and failures, availability of nodes, which varies both in time and space, and their lack of willingness to participate in the learning process. Chapter 4 will investigate the latter problem and will present a method based on game theory to incentivize nodes to cooperate in the training within the context of a Decentralized Learning scenario.

As previously highlighted, access to vast datasets is one of the driving factors of ML success. Despite the massive amount of data produced continuously by a wide range of devices, significant attention has been dedicated to synthetic samples created with Generative AI (GenAI), which has recently emerged as a powerful technology, capturing significant interest in both research and industry. Synthetic data produced by such generative models have been proven to enhance the performance of DNNs [10]

and can be employed to reduce the burden, and thus the resources allocated and costs, for real data collection [11]. In some scenarios, collecting extensive real data could be difficult, unsafe or costly, such as acquiring crash accident data, which could help improve autonomous driving. In such contexts, synthetic data would serve as valuable resources for developing, training and testing ML algorithms.

Driven by these motivations, as comprehensively explained in the next section, in the second part of this manuscript we will implement and evaluate a generative model to produce synthetic novel views of a given object; finally, we will demonstrate the benefits of using such generated images for classification tasks.

The remainder of this chapter will highlight the main contributions and introduce the research questions addressed in this work, followed by a detailed outline of the thesis.

1.1 Main contributions and research questions

The first main contribution of this thesis is the design and development of solutions to address two of the main challenges of the CL paradigm, namely the energy consumption optimization and the need for an incentive mechanism to foster collaboration, as nodes might not want to share their resources to participate in the training stage, unless it is beneficial for them. The second main contribution focuses on the generation – by means of generative models – of high-quality synthetic data, which are proven to enhance the final accuracy and the resilience of classification systems.

1.1.1 Cooperative Learning

Minimization of training energy cost

Training in CL scenarios is influenced by many factors and decisions about them depend both on the objective and constraints. In particular, we aim to minimize the energy entailed by the cooperative training, considering both time and learning quality requirements. Notably, our focus is on optimizing efficiency rather than maximizing performance, which is the primary goal of many other concurrent works; our objective, thus, is not necessarily to achieve the best results, but to ensure that the model reaches a minimum level of performance, namely, a test loss lower than

a target threshold value. In this context, we consider two different CL scenarios. In the first one, the DNN layers, or multiple instances thereof, are assigned and trained at different nodes. In the second scenario, sets of nodes sequentially train a model, which can be pruned throughout the learning. Pruning is a widely used compression technique for DNNs that removes the model's parameters with a lower impact on performance, thereby reducing model complexity. A great advantage of pruning is that it allows resource-constrained nodes, i.e., with limited computational capabilities and memory, to contribute to the learning. Contrary to the first scenario where nodes train only one or a few layers, in this second case every node trains all the layers of the model, whether full or pruned.

In detail, we answered the following research questions (RQs).

RQ1: How to efficiently train a DNN whose layers, or multiple instances of them, are run at different nodes that cooperatively perform a distributed learning task?

More in detail, how to make decisions about data and model structure, i.e., the DNN layer instances to employ, and how to map them with the physical nodes in order to minimize the training energy costs while meeting performance and time constraints?

To answer RQ1, due to the interdependency among such factors, we highlight the need for the learning controller to make the abovementioned decisions in a joint manner. In other words, the model structure, the instance-node mapping and the resources are mutually adapted. Then, we characterize the learning process and we design the RightTrain framework: given the set of possible DNN structures ordered by their computational requirements, it starts with the least demanding configuration, maps the layer instances to nodes and adjusts the allocated data and resources to improve efficiency. This procedure is iteratively performed until a feasible solution is found. Notably, the mapping is performed with a delay-aware Steiner tree computed on top of the expanded-graph that represents the system. RightTrain solves the problem, which is NP-hard, in polynomial time and yields near-optimal solutions.

This part is extensively presented in Chapter 2. Our contribution on this topic can be found in:

1. Malandrino, F., Chiasserini, C. F., **Di Giacomo, G.** (2022, March). Energy-efficient Training of Distributed DNNs in the Mobile-edge-cloud Continuum.

In 2022 17th Wireless On-Demand Network Systems and Services Conference (WONS) (pp. 1-4). IEEE

2. Malandrino, F., Chiasserini, C. F., **Di Giacomo, G.** (2022). Efficient distributed DNNs in the mobile-edge-cloud continuum. *IEEE/ACM Transactions on Networking*, 31(4), 1702-1716

RQ2: How to fine-tune DNN model compression to available nodes and resources for efficient training in Cooperative Learning?

Specifically, how to make decisions about the models – full or pruned versions – to use, when to switch among them and which nodes, hence data, to use in order to minimize energy consumption during training subject to quality constraints and time limits?

To answer RQ2, we develop and evaluate an algorithmic solution called PACT. PACT is executed at the learning orchestrator and makes joint decisions about the components characterizing the cooperative learning, by leveraging (i) a time-expanded graph to represent the states and the possible actions of the training process and (ii) an approximate dynamic programming approach that employs Neural Network estimators for the prediction of the test loss values. Importantly, we can adjust the graph's granularity to balance its size and PACT performance. PACT has polynomial worst-case complexity, as opposed to the NP-hardness of the problem, and allows us to outperform alternative solutions, achieving near-optimal results.

This part will be comprehensively presented in Chapter 3. Our work on this subject has been published in:

1. Malandrino, F., **Di Giacomo, G.**, Karamzade, A., Levorato, M., Chiasserini, C. F. (2023, May). Matching DNN compression and cooperative training with resources and data availability. In IEEE INFOCOM 2023-IEEE Conference on Computer Communications (pp. 1-10). IEEE

2. Malandrino, F., **Di Giacomo, G.**, Karamzade, A., Levorato, M., Chiasserini, C. F. (2023). Tuning DNN Model Compression to Resource and Data Availability in Cooperative Training. *IEEE/ACM Transactions on Networking*, 32(2), 1600-1615

Incentive mechanisms for cooperation

We also focus on another relevant challenge in CL: nodes may lack motivation to participate in the training process, as this entails non-negligible computational, communication and, hence, energy costs. In detail, we have addressed the following research question.

RQ3: In Decentralized Learning systems, how can cooperation be fostered among selfish nodes that cannot be forced to contribute to the training?

Specifically, how can we achieve effective and fair cooperation considering heterogeneous nodes with diverse computational resources and different models to train?

To answer RQ3, we propose GENIAL, a game-theoretic approach, based on the popular Generous Tit-for-Tat strategy. Through both theoretical analysis and experimental results, we show the benefits for nodes of adopting a generous behavior, where they cooperate even if it is not immediately beneficial to them. GENIAL yields a Nash equilibrium, which also converges to the Pareto optimal operating point, achieving fairness among nodes.

Our research on this topic, along with the evaluation of our solution, is covered in Chapter 4. The findings from these studies have been published in the following conference paper:

1. **Di Giacomo, G.**, Malandrino, F., Chiasserini, C. F. (2024, June). Generosity Pays Off: A Game-Theoretic Study of Cooperation in Decentralized Learning. In 2024 IEEE International Conference on Communications Workshops (ICC Workshops) (pp. 105-110). IEEE.

1.1.2 Generation and effectiveness of synthetic data

This branch of work focuses on an innovative research area, namely the generation of synthetic data using generative models. In the previous section we have highlighted their utility and effectiveness: driven by these motivations, we consider a multi-view scenario where an object or a scene can be observed from multiple points of view, which can offer richer insights with respect to observations from a single perspective. In such a context, however, some views may not be available, e.g., for sensor failure, and it could be necessary to reconstruct them by employing a generative model. To

this end, we focus on diffusion models, a specific class of generative models that represent the state-of-the-art in data generation.

In detail, we addressed the following research questions.

RQ4: How to generate high-quality synthetic data images in a multi-view scenario?

In particular, given a set of views and assuming some of them to be missing, how can we generate them conditioned on the available views?

To address this question, we develop DiMViS, a joint diffusion model for Novel View Synthesis that leverages a masked diffusion process specifically designed to enable conditional generation capability, accommodating a variable number of both input and output views. Furthermore, the model is fine-tuned starting from a pre-trained state-of-the-art model. Combined, these two factors allow the model to outperform the competitors by up to 15% in perceptual similarity between ground truth and generated images, as demonstrated by our quantitative evaluation.

More details about this work are extensively presented in Chapter 5. Our research on this subject has been published in the following conference paper:

1. **Di Giacomo, G.**, Franzese, G., Cerquitelli, T., Chiasserini, C. F., Michiardi, P. (2024). DiMViS: Diffusion-based multi-view synthesis. In ICML 2024 Workshop on Structured Probabilistic Inference & Generative Modeling

RQ5: How can we use a diffusion model for Novel View Synthesis to enhance the performance of image classification systems?

Additionally, is it beneficial to leverage synthetic views at inference time?

To answer RQ5, we propose DiMViDA, a Diffusion-based Multi-View Data Augmentation technique for image classification. In particular, we use the novel views produced by the diffusion model to augment the training set. Furthermore, DiMViDA employs synthetic views also at inference. Specifically, starting from one available image of the object to classify, we produce novel synthetic views thereof; eventually, we combine in a straightforward yet effective manner the true and the synthetic views for the final classification. Our experimental campaign demonstrates the effectiveness of our approach, showing increased accuracy with two CNNs up to 20% when training them from scratch and, to a lesser extent, when fine-tuning

the models. Notably, the proposed inference scheme can increase the reliability of a classification system where the inference task is offloaded to an edge node. The latter, indeed, may not always receive the complete set of views required for the classification due to failures: thus, it could use the diffusion model to produce the missing views, and use them with the real available data to achieve a higher accuracy, hence making the system more resilient.

This work is described in detail in Chapter 6. Our research on the topic has been published in the following conference paper:

1. **Di Giacomo, G.**, Franzese, G., Cerquitelli, T., Chiasserini, C. F., Michiardi, P. (2024). DiMViDA: Diffusion-based Multi-View Data Augmentation. In IEEE CAMAD 2024. IEEE

1.2 Outline

The rest of this manuscript is structured as follows.

- **Chapter 2** considers a distributed learning task in the mobile-edge-cloud continuum. First, it presents the factors that affect the distributed training, namely, data, DNN layers composition and layer-node mapping. Then, it defines the problem of minimizing energy consumption subject to performance and time constraints, with the learning controller required to make joint decisions about the above-mentioned factors, due to their mutual dependencies. After characterizing the time and energy entailed by the training, it uses an auxiliary DNN to estimate the number of required epochs depending on the model structure. Then, it presents RightTrain, an algorithmic framework with polynomial complexity that achieves solutions close to the optimum. Finally, the proposed solution is validated by implementing it on a testbed.
- **Chapter 3** focuses on a cooperative learning scenario, where nodes, or set thereof, cooperate to sequentially perform a training task. Importantly, the model is pruned during the training, introducing new challenges in addition to the well-known ones, such as the selection of the nodes to use. Specifically, the learning orchestrator also needs to decide when and how much to prune the model during the training. Chapter 3 presents the model systems and defines

the problem of making joint decisions aimed at minimizing energy consumption during training, while satisfying two constraints, namely a maximum time limit and a minimum performance threshold to reach. To achieve this, it introduces PACT, an algorithm based on an approximate dynamic programming approach that leverages the estimation of the loss during the learning process. Finally, the proposed solution is proven to make near-optimal decisions in polynomial time, outperforming alternative methods.

- **Chapter 4** presents GENIAL, a game-theoretic incentive method to foster cooperation among learning nodes during training in decentralized settings. GENIAL is based on the Generous Tit-for-Tat strategy and takes into account the node heterogeneity in terms of computational capabilities, complexity of models to train and dataset size. GENIAL yields Nash equilibrium and Pareto optimality and its effectiveness is demonstrated by numerical experiments.

- **Chapter 5** introduces DiMViS, an innovative generative diffusion-based model for Novel View Synthesis. Differently from other methods, DiMViS enables the generation of multiple images, conditioned on a variable number of reference views, by leveraging a joint diffusion model to learn the conditional data distribution. DiMViS integrates a masked diffusion approach on top of the pre-trained Stable Diffusion model, outperforming state-of-the-art competitors, as shown by our experimental evaluation.

- **Chapter 6** presents DiMViDA, a Diffusion-based Multi-View Data Augmentation method for image classification. The proposed approach leverages a Novel View Synthesis model similar to the one introduced in Chapter 5 to augment both the training and the inference pipelines, resulting in enhanced accuracy. Notably, it shows how the proposed inference method can be used in an edge-assisted classification system to increase its resilience.

- **Chapter 7** provides a summary of the presented work, emphasizing the key contributions and the results achieved. Lastly, it presents an overview of the future research direction and highlights the open challenges.

Chapter 2

Efficient Distributed DNNs in the Mobile-edge-cloud Continuum

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

Part of the work described in this chapter has already been published in Malandrino, F., Chiasserini, C. F., **Di Giacomo, G.** (2022). Efficient distributed DNNs in the mobile-edge-cloud continuum. *IEEE/ACM Transactions on Networking*, 31(4), 1702-1716.

2.1 Introduction

Enabling technologies like 5G networks and distributed machine learning (ML) have fostered the emergence of the so-called *Internet of Intelligent Things* networking paradigm, allowing user equipment (UEs), e.g., smartphones or smart-city actuators, to leverage cloud-based artificial intelligence services. This scenario is expected to further evolve towards the edge intelligence paradigm [12, 13]: ML-based applications will move from remote, cloud-based servers to the mobile network, including computation-capable devices at the network edge and mobile UEs. Indeed, recent reports [14] highlight how the capability of edge and mobile devices is growing much faster than cloud ones, soon leading to complete interoperability among mobile, edge, and cloud and, thus, to the formation of a *continuum*.

The ML tasks to perform will be as diverse as the devices performing them. Indeed, the best ML approach to adopt depends upon such factors as the application and scenario at hand, as well as the type and quantity of available data. Possible options include supervised [15–17] and unsupervised learning (most notably, deep domain adaptation (DDA) [18–20]), as well as hybrid applications combining labeled and unlabeled data through multiple learning and pseudo-labeling techniques [21]. Many of these tasks can be accomplished through deep neural networks (DNNs), built by combining a sequence of *layers* of different types. The DNNs used for unsupervised and hybrid learning tend to be more complex than their supervised learning counterparts, however, they use the very same building blocks, e.g., convolutional and fully-connected layers [15, 17]. The possibility of using a relatively small set of building blocks to support a wide set of ML-based applications has motivated the ML-as-a-Service (MLaaS), whereby the network provides a set of *customized* ML-based services, e.g., image recognition or clustering, to the applications using the network resources.

Owing to the complexity of the learning tasks to perform, as well as to the need to keep as much as possible the information coming from different sources local, it is expected that most of MLaaS learning will be performed in a *distributed* fashion. Distributed learning is an excellent fit for edge intelligence scenarios, as it envisions leveraging the data and resources of multiple nodes in order to perform a common learning task. Additionally, recent works on ML techniques tailored for lower-powered devices [22, 23] has significantly extended the types of devices that can be leveraged for distributed ML, along with their diversity. We thus argue that ML-based services

can seamlessly use any of the nodes of the mobile-edge-cloud continuum, so as to ensure that requirements (e.g., learning time) are met.

Among the factors influencing the performance of MLaaS, combined with distributed learning, the most prominent are (i) the quantity and diversity of the input data used for training; (ii) the actual learning strategy employed, e.g., the layers composing the DNN; (iii) the resources, (e.g., computational) available to the learning task. Such decisions strongly impact one another, e.g., using more data requires more resources to promptly process them; thus, it is important that they are made in a joint manner. The split learning (SL) paradigm [24–26], envisioning to run a subset of the DNN layers at each level of the network topology, represents a significant step in this direction. However, SL is concerned only with placement decisions and, since it only splits DNN into as many parts as there are network topology layers (e.g., one for edge and one for cloud), may not always be able to reap the full benefits of the mobile-edge-cloud continuum.

Another limit of state-of-the-art works on distributed ML is their emphasis on learning *effectiveness*, e.g., classification accuracy, over *efficiency*. Indeed, most existing approaches are concerned with locating and selecting the highest possible quantity of computational resources, and use them to process the largest possible quantity of data, in order to obtain the highest-quality possible learning. However, awareness is rising that seeking the utmost performance is not necessarily the most desirable strategy in real-world situations, and the concept of ML efficiency – as opposed to sheer performance – is rapidly gaining prominence [27]. In fact, while *inference* has a relatively minor impact on the total resource consumption, the *training* of a DNN is very demanding in terms of processing power, hence, energy.

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

More specifically, our contributions can be summarized as follows:

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

As discussed in Sec. 2.2, several existing works have tackled the problem of selecting the computational and network resources needed for a given learning task, and a few have studied the impact of different DNN structures on the learning performance. Our work is, however, the first to identify and solve the important challenge of *adapting* the DNN structure, the network resources and the size of the datasets to one another, thus achieving unparalleled learning efficiency and performance.

The remainder of this chapter is organized as follows. After discussing related work in Sec. 2.2 and summarizing our main results in Sec. 2.3, we describe our system model and problem formulation in Sec. 2.4 and our characterization of learning performance in Sec. 2.5. We then introduce the RightTrain solution concept in Sec. 2.6, and formally prove its complexity and competitive ratio properties in Sec. 2.7. Finally, Sec. 2.8 shows the performance of RightTrain against SL, Sec. 2.9 reports how we validate our model and approach through a lab test-bed, and Sec. 2.10 concludes the chapter.

2.2 Related Work

A first body of works related to ours [28–30] target the problem of characterizing the performance of distributed ML, accounting for such aspects as the topology of the *logical* network formed by cooperating nodes [28] and the computational [29, 30] and communication [30] resources they are assigned.

Concerning distributed learning techniques themselves, one of the most prominent is federated learning (FL) [31]. FL has become one of the most popular approaches to cooperative learning also in mobile [32] and MEC-based [29] scenarios. Since nodes are expected to contribute their own resources to learning, incentive mechanisms may be necessary to foster cooperation [33], also employing blockchain [34].

Several recent works have endeavored to characterize and improve the performance of FL under realistic conditions, most notably, learning nodes with heterogeneous datasets and/or capabilities. The authors of [35] consider the classic FedAvg strategy, and characterize its performance, e.g., the loss reduction as a function of the number of local epochs and global iterations. The later work [36] aims at going beyond FedAvg and proposes an alternative strategy called FedUN. FedUN optimizes the convergence speed by choosing ("sampling") the learning nodes to use at each epoch – and weighting their updates – based upon local gradient information. [36] also provides a lower bound for FedUN's loss reduction at each epoch, hence, the total convergence time. [37] takes a different approach and tackles the issue of fairness, i.e., how to ensure that devices with different capabilities and/or datasets have similar learning performance. The strategy envisioned in [37] is predicated upon giving *more* weight to updates from the nodes with the worst performance, resulting in a better learning quality for such nodes at the cost of a higher number of global iterations.

Split learning (SL) is a recently-emerging paradigm predicated on partitioning the DNN among the nodes participating in the learning process. SL drops FL's requirement that all nodes have the same DNN, and has been found to outperform ML in a wide variety of scenarios [26], owing to its ability to match the learning operations and the hardware performing them. Other works envision similar approaches, based on choosing the right network node to run each layer of a DNN and accounting for device capability [38], network latency [38, 39], and privacy [38, 40]. [39] takes a further step, combining DNN splitting with "right-sizing", i.e., removing some DNN layers if they are not necessary to reach the required learning quality. However,

that work only focuses on inference, and neither the quantity of data to use nor the resources to assign are accounted for.

Among the few works jointly making learning- and network-related decisions, [41] aims at (i) right-sizing the DNN for the task at hand, i.e., skipping some layers if the classification precision is sufficiently high, and (ii) offloading a part of the DNN to edge-based nodes. Notice, however, that [41] only supports DNNs with a chain topology, and does not support the use of multiple sources of information. Still in the context of inference, the authors of [42] envision partitioning the DNN into an arbitrary number of parts, and running each of them at the most appropriate node in the edge-cloud continuum. In a similar setting, [43] addresses security issues, placing different layers of an image-classification DNN on different devices, preventing any device from seeing enough layers to reconstruct the original image.

Recent work [44] targets a heterogeneous scenario similar to the mobile-edge-cloud continuum, and envisions an *inference delivery network* whereby each node offers one or more ML models. Inference tasks – which are assumed to require one of several alternative models, e.g., DNNs with different architectures – can then be carried out at the most appropriate node, accounting for both cost and delay issues. Compared to [44], our work (i) targets the *learning* phase, which is the most challenging and resource-intensive; (ii) allows breaking down *one* model (e.g., one DNN) across multiple devices, and (iii) does not depend upon the assumption that alternative models exist for a given task.

Our work is also related to the recent but growing body of works accounting for ML energy consumption, and the resulting carbon footprint. As reported in [45], training one complex ML model may lead to a carbon footprint equivalent to 5 times the lifetime emissions of an average car. Thus, it is critical to envision solutions that, exploiting the edge intelligence concept [13, 46], effectively exploit the physical proximity of a large number of devices, each collecting data and equipped with computational and memory resources. At a higher level, as advocated in [27], this calls for a different view of ML goals where the focus shifts from sheer learning quality (e.g., classification accuracy) to the more comprehensive concept of *efficiency*.

In this work, we also address a placement and resource allocation problem, which is tackled by many works, such as [47–51], as a service graph optimization problem.

Finally, a preliminary version of RightTrain has been presented in our conference paper [52].

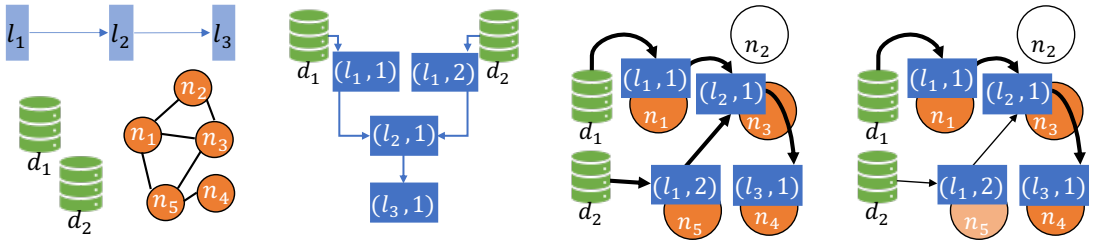
Novelty. Our holistic approach contributes to making ubiquitous ML reality, *jointly* addressing issues that earlier have been only marginally or incidentally dealt with. Specifically, we account for the mutual influence of the decisions on (i) the data used for DNNs training, (ii) the DNN structure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

2.3 Main Results and Roadmap

Our first major contribution is represented by the system model and problem formulation summarized in Fig. 2.1 and described in Sec. 2.4. The system model accounts for all the main entities and aspects involved in distributed training of DNNs over the mobile-edge-cloud continuum. It can describe ML approaches leveraging data parallelism, model parallelism, or a combination of both – including split learning [24, 25]. The problem formulation then formalizes how the decisions of (i) selecting the input data to be leveraged for learning, (ii) choosing the DNN structure to be used, and (iii) matching DNN layers with physical servers influence each other, and determine the learning efficiency under the constraints imposed by both the learning process and the network system. The problem is shown to be NP-hard.

Next, through a combination of existing measurements taken from the literature and our own experiments, we assess how such decisions impact the learning performance, namely, learning time, learning quality, and energy consumption (Sec. 2.5). Importantly, so doing, we bridge the gap between the abstract system model of Sec. 2.4 and actual, real-world distributed ML solutions.

Building upon the above results and in light of the problem complexity, we envision a solution concept, also summarized in Fig. 2.3. Its main component is the RightTrain algorithm, detailed in Alg. 1 (Sec. 2.6), which leverages expanded graphs such as the one shown in Fig. 2.4 and applies a delay-aware Steiner tree on such graph, to make near-optimal decisions on data selection, DNN structure, and layer-to-node matching. More specifically, as proven in Sec. 2.7, our solution strategy has polynomial worst-case time complexity (Property 3) and a competitive ratio of $2(1+\epsilon)$. Finally, our performance evaluation shows that the proposed solution



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

Fig. 2.1 The different stages of the RightTrain approach. (a): input data, namely, a set of DNN *layers* (light blue), a set of *data sources* (green), and a *physical graph* composed of physical nodes (orange). (b): an instance tree, whose nodes are data sources and *layer instances*. (c): a possible *deployment*, associating layer instances to three out of the five physical nodes. (d): a *refined* deployment, using only some of d_2's data and, accordingly, reducing the computational power allocated to layer instance $(l_1, 2)$.

reduces by 50% the energy consumption of a learning task when compared to SL, while our lab test-bed implementation shows its feasibility.

2.4 System Model and Problem Formulation

Our system model describes a DNN training task leveraging distributed learning, and exploiting the resources of multiple mobile, edge, and cloud nodes (hereinafter also referred to as learning nodes). Such nodes are coordinated by a *central controller*, typically running at the edge of the network infrastructure, which can communicate with all learning nodes and collects information on their capabilities and position. The entities the model represents, along with the decisions the central controller has to make, are depicted in Fig. 2.1.

2.4.1 Input information

Under the DNN paradigm, learning tasks are performed by a set of *layers* of different types (e.g., fully-connected or convolutional), organized as a *tree*: the learning result is the output of the tree root, while leaves correspond to data sources. Each layer has a local set of *parameters* that define its behavior, e.g., the weights of a fully-connected layer, and *training* a DNN means finding the parameter values that minimize a global error function. As an example, for a classification task, the learning output $\mathbf{y}$

represents the probabilities associated with each class, and a typically-used loss function is cross-entropy, defined as $f(\hat{\mathbf{y}}, \mathbf{y}) = \hat{\mathbf{y}}^{\mathsf{H}} \log \hat{\mathbf{y}}$, where $\hat{\mathbf{y}}$ is a column vector containing the ground truth, and $^{\mathsf{H}}$ represents the transpose operator. Training proceeds in an iterative fashion [53] through several *epochs*, each including (i) a *forward pass*, where the input data traverses all instances from the leaves to the tree root, and (ii) a *backward pass* where gradients follow the opposite route and local parameters at each layer are adjusted so as to reduce the global loss function. The most commonly-employed optimization algorithm is stochastic gradient descent (SGD), though alternatives tailored to ML have been proposed as well [54].

Thus, the input to our problem includes (see Fig. 2.1(a)):

- a set $\mathcal{L} = [l_1, \ldots, l_L]$ of *DNN layers* connected to each other to form the DNN structure to implement;
- a set $\mathcal{N}$ of *physical nodes*, i.e., mobile, edge or cloud [14] nodes with the computational capability to run one (or more) layer instances;
- a set $\mathcal{D}$ of *data sources*, which may be colocated with physical nodes.

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

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

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

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

2.4.2 Decision variables

The main decisions to make concern (i) how many layer instances to create and how to connect them, as exemplified in Fig. 2.1(b), (ii) how to deploy the instances onto the physical nodes, as shown in Fig. 2.1(c), and (iii) how to assign the computational and network resources, as per Fig. 2.1(d).

Layer instances and instance trees. For each layer $l \in \mathcal{L}$, we shall create at least one and at most $\alpha|\mathcal{D}|$ *layer instances*, with $\alpha \geq 1$ being a redundancy factor. Notably, the possibility of deploying multiple instances of a layer allows for greater flexibility in accommodating multiple data sources. Each layer instance runs at a physical node, and it is identified as a pair (l, i), where i is an index ranging from 1 to $\alpha|\mathcal{D}|$, while the set of instances is denoted by $\mathcal{I}$. As shown in Fig. 2.1(b), layer instances and data sources in $\mathcal{D}$ are connected to form an *instance tree*, with binary variables $y(l, i, m, j) \in \{0, 1\}$ expressing whether layer instance (l, i) shall be connected to layer instance (m, j). As shown in Fig. 2.1(b), data sources $d \in \mathcal{D}$ are part of the instance tree, however, each can only be associated with at most one instance $(d, 1)$. Associating zero instances with a certain data source means not using it, e.g., because a sufficient quantity of data is already available.

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

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

$$\chi(l, i, m, j) = \begin{cases} x(l, 1, m, j) & \text{if } l{\in}\mathcal{D} \wedge i{=}1, \\ q(l) \sum_{(h,k)} \chi(h, k, l, i) & \text{otherwise.} \end{cases} \tag{2.1}$$

2.4.3 Constraints

The decision variables $y(l,i,m,j)$, $z(l,i,n)$, $\rho(l,i)$, and $x(d,1,l,i)$ are subject to several constraints. Two of them concern the instance tree exemplified in Fig. 2.1(b) and expressed by the y-variables. Specifically, we must deploy at least one instance of each layer, i.e.,

$$\sum_{i\in[1\ldots\alpha|\mathcal{D}|]}\sum_{(m,j)\in\mathcal{I}} y(l,i,m,j) \geq 1, \quad \forall l \in \mathcal{L}. \tag{2.2}$$

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

$$y(l,i,m,j) \leq \mathbb{1}_{[l \text{ is child of } m]}, \tag{2.3}$$

Then, moving to the deployment decisions exemplified in Fig. 2.1(c) and expressed by the z-variables, we must ensure that each instance is deployed at exactly one physical node:

$$\sum_{n\in\mathcal{N}} z(l,i,n) = 1, \quad \forall (l,i) \in \mathcal{I}. \tag{2.4}$$

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

$$z(l,i,n) \leq \mu(l,n), \quad \forall (l,i) \in \mathcal{I}, n \in \mathcal{N}, \tag{2.5}$$

where we assume that each node statically allocates memory for the layers in advance. As for the computational resource allocation and data exchange decisions exemplified in Fig. 2.1(c) and expressed by ρ and x-variables, we must ensure that the total amount of resources allocated to all the instances running at each node n does not exceed the available one, i.e.,

$$\sum_{(l,i)\in\mathcal{I}:\ \nu(l,i)=n} \rho(l,i) \leq R(n), \quad \forall n \in \mathcal{N}. \tag{2.6}$$

$$\sum_{\substack{(l,i):\ \nu(l,i)=n \\ (m,j):\ \nu(m,j)=n'}} \chi(l,i,m,j) \leq y(l,i,m,j)S(n,n') \quad \forall n,n' \in \mathcal{N}. \tag{2.7}$$

Finally, we enforce generalized flow conservation [55], i.e., the quantity of data going out of layer instance (l,i) cannot exceed the product between the quantity of

incoming data and $q(l)$:

$$\sum_{(m,j)\in\mathcal{I}} \chi(l,i,m,j) \leq q(l) \sum_{(g,h)\in\mathcal{I}} \chi(g,h,l,i) \quad \forall (l,i) \in \mathcal{I}. \tag{2.8}$$

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

$$\sum_{(l,i)\in\mathcal{I}} x(d,1,l,i) \leq \Delta(d), \quad \forall d \in \mathcal{D}. \tag{2.9}$$

2.4.4 Objective function

Decisions x, y, z and ρ fully describe the behavior of the distributed learning application. However, they do not directly express: (i) the time taken by each learning epoch, (ii) the energy consumed by each learning epoch, and (iii) the number of epochs needed to attain the required loss function value $\epsilon^{\max}$. We account for these quantities through functions $\mathbf{T}(x,y,z,\rho)$, $\mathbf{E}(x,y,z,\rho)$, and $\mathbf{K}(y,\epsilon)$, respectively. The first two are described in Sec. 2.5.1, while the third one in Sec. 2.5.2.

Given functions $\mathbf{T}$, $\mathbf{E}$, and $\mathbf{K}$, we formulate our problem as minimizing the learning energy consumption, subject to (2.2)–(2.9) and to achieving the required loss $\epsilon^{\max}$ by time $T^{\max}$:

$$\min_{x,y,z,\rho} \mathbf{K}(y,\epsilon^{\max})\mathbf{E}(x,y,z,\rho) \tag{2.10}$$

$$\text{s.t. } (2.2)-(2.9), \ \ \mathbf{K}(y,\epsilon^{\max})\mathbf{T}(x,y,z,\rho) \leq T^{\max}. \tag{2.11}$$

As we will formally prove in Sec. 2.7, the above problem is NP-hard.

2.5 Characterizing the Learning Performance

We now show how the performance of the learning process can be characterized, with reference to the time and energy it takes to perform one epoch (Sec. 2.5.1), and the number of epochs necessary to achieve the required learning quality (Sec. 2.5.2).

2.5.1 Epoch duration and energy consumption

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

$$t^{l,i}_{\text{comp}} = \frac{r(l)}{\rho(l,i)} \sum_{(k,h)\in\mathcal{I}} \chi(k, h, l, i)\,. \tag{2.12}$$

As per (2.12), the computation time is given by the ratio between the number of operations to perform (e.g., given by the amount of data to process times the requirement $r(l)$) and the quantity $\rho(l, i)$ of computing resources assigned to that instance. Note that, by constraint (2.7), the sum in (2.12) accounts for all the data transferred from the children instances k to the parent instance l. We also define the network time, $t^{n,n'}_{\text{net}}$, needed to transfer data from node n to node n', which depends upon the quantity of data transferred from the children instances k running at n to the parent instance l running at n':

$$t^{n,n'}_{\text{net}} = \frac{\sum_{(h,k),(l,i)\in\mathcal{I}:\, \nu(h,k)=n, \nu(l,i)=n'} \chi(h, k, l, i)}{S(n, n')}\,. \tag{2.13}$$

Then, we can compute times $t^{i,j}_{\text{begin}}$ and $t^{i,j}_{\text{end}}$ at which instance (l, i) starts and ends its computation. Specifically, each instance can only start its processing when all the data it needs from preceding nodes has arrived [28], while its end time is given by the sum between the layer instance begin time and computing time:

$$t^{l,i}_{\text{begin}} = \max_{(h,k)\in\mathcal{I}} \left[y(h, k, l, i) \left(t^{h,k}_{\text{end}} + t^{\nu(h,k),\nu(l,i)}_{\text{net}} \right) \right], \tag{2.14}$$

$$t^{l,i}_{\text{end}} = t^{l,i}_{\text{begin}} + t^{l,i}_{\text{comp}}\,. \tag{2.15}$$

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

$$\mathbf{T}(x, y, z, \rho) = \max_{(l,i)\in\mathcal{I}:\, l=L} t^{l,i}_{\text{end}}\,. \tag{2.16}$$

Estimating the energy consumption associated with ML computational tasks has been the focus of a significant body of research [56]. In general, the energy consumption associated with a task (in our case, a learning instance running at a node) is determined by its usage of CPU, memory, and GPU resources [57, Eq. (1)]. Those, in turn, depend upon the layer implemented, the quantity of data it processes, and the characteristics of the node itself. Thus, for each layer instance $(l,i) \in \mathcal{I}$, we can write:

$$E^{l,i}_{\text{comp}} = t^{l,i}_{\text{comp}} \left[e_{\text{p}}(\nu(l,i))\rho(l,i) + e_{\text{f}}(l,\nu(l,i)) \right]. \tag{2.17}$$

In (2.17), we recall that $t^{l,i}_{\text{comp}}$ is the computation time for each layer instance at each epoch. Such a time is multiplied by the power consumed by the node $\nu(l,i)$ at which the instance runs, which depends upon the quantity of resources assigned to it. Parameters $e_{\text{p}}(n)$ and $e_{\text{f}}(l,n)$ express, respectively, the power consumed at node n to provide one unit of CPU, and the power consumed at that node to support the memory and storage requirements of an instance of layer l. Both quantities are parameters for our model and can be set following the methodology introduced in [58], i.e., analyzing the requirements of individual layers and the resulting energy consumption. Furthermore, each instance (l,i) running at node $\nu(l,i) \in \mathcal{N}$ implies additional energy consumption due to data transmissions over the network:

$$E^{l,i}_{\text{net}} = \sum_{(m,j)\in\mathcal{I}} \left[e_{\text{net}}(\nu(l,i),\nu(m,j))\chi(l,i,m,j) \right]. \tag{2.18}$$

In (2.18), the energy spent for data transmissions for layer instance $(l,i) \in \mathcal{I}$ is given by the sum of the energy contributions of all transmissions of data going from instance $(l,i) \in \mathcal{I}$ to any other layer instance $(m,j) \in \mathcal{I}$. Each contribution is determined by the quantity of data sent from (l,i) to (m,j) multiplied by the factor $e_{\text{net}}(\nu(l,i),\nu(m,j))$, expressing how much energy is required to transmit one unit of data from node $\nu(l,i)$ to node $\nu(m,j)$.

The energy consumed during one epoch can be found summing the instance-specific energy consumption values (2.17):

$$\mathbf{E}(x,y,z,\rho) = \sum_{(i,l)\in\mathcal{I}} \left(E^{l,i}_{\text{comp}} + E^{l,i}_{\text{net}} \right). \tag{2.19}$$

Finally, $\mathbf{E}(x, y, z, \rho)$ can be mapped into *carbon* emissions following the methodology in [59], which accounts for the quantity of CO_2 emitted for each kilowatt-hour of consumed energy.

Note that, for simplicity, our analysis has considered only the forward pass. However, our analysis is still valid as we assume that the computational energy is the dominant component, i.e., the transmission energy is negligible; therefore, as the computational energy required for the forward pass is proportional to that of the backward, and hence, of the entire epoch, the objective (2.10) is not affected. As for the constraint, accounting for the backward pass duration simply requires applying the same procedure explained above.

2.5.2 Overall learning time and energy consumption

In Sec. 2.5.1, we have derived the time $\mathbf{T}$ and energy $\mathbf{E}$ required by each epoch as functions of our decision variables. In order to obtain the total time and energy required by the overall learning process, we need to multiply such values by the number $\mathbf{K}(y, \epsilon^{\max})$ of epochs needed to attain the target learning quality $\epsilon^{\max}$. Deriving a closed-form expression for the number $\mathbf{K}$, in a manner similar to $\mathbf{T}$ and $\mathbf{E}$, is currently possible only for few, simple scenarios among those that we address. Indeed, as discussed in Sec. 2.2, currently-available results only target scenarios where *all* nodes share the same DNN, and *all* layers of said DNN are averaged, i.e., *either* data *or* model parallelism are employed, but not a combination of both. Thus, to show the ability of our approach to deal with arbitrary – and potentially more efficient – instance trees and assignment decisions, we use an auxiliary ML model to estimate $\mathbf{K}$, owing to the ML ability to reveal and leverage the structure underlying those phenomena that are too complex to describe through a traditional model.

A key observation we make is that, if we assume to always use all the available data, i.e., to honor constraints (2.8) and (2.9) with the equality sign, then the number $\mathbf{K}$ of epochs to run only depends upon the instance tree we consider, i.e., the $y(l, i, m, j)$ variables. Given such decisions, as well as the target learning quality $\epsilon^{\max}$, our approach follows the steps set forth below:

(i) we run a number M of experiments, each for different values of the y variables;

(ii) for each experiment, we determine the resulting value of $\mathbf{K}(y, \epsilon^{\max})$;

(iii) using the above information, we train an *auxiliary* DNN that can predict the value of $\mathbf{K}(y, \epsilon^{\max})$, given arbitrary values for the y variables.

A similar approach has been used, among others, in [60] for mmWave-based vehicular networks, and in [61] for optical networks.

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

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

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

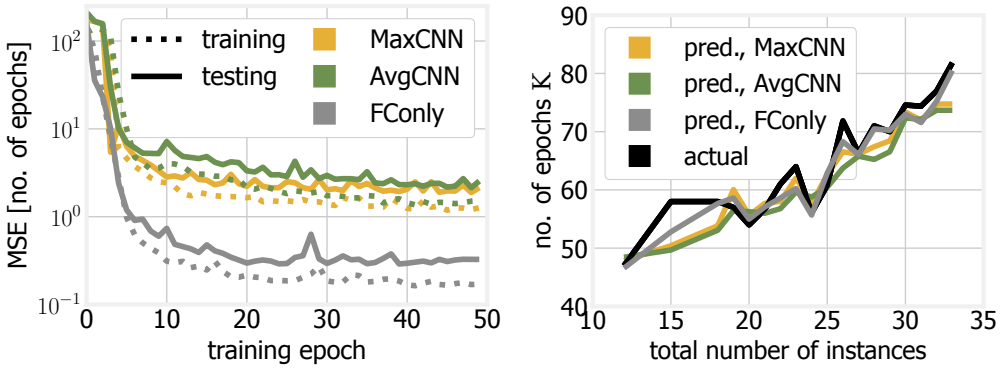


Fig. 2.2 Auxiliary DNN for estimating the number of required epochs: resulting MSE (left), and real and predicted values of the number K of iterations (right).

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

We validate our approach using an image classification task leveraging the AlexNet DNN [2] and the CIFAR-10 [62] dataset. The results are summarized in Fig. 2.2(left), depicting the evolution of the MSE across training epochs. We can observe that, for all the auxiliary DNN architectures, both training and testing MSE values rapidly converge to very small values, below 1 for the FConly architecture. This highlights how our approach is indeed very effective in predicting the number K of iterations required for convergence. Fig. 2.2(right) further shows that, consistently with [28], K tends to grow as the number of layer instances increases, and the FConly architecture is always associated with very close predictions.

Since the applicability of our methodology to real-world cases hinges on the availability of sufficient training data, it is worth highlighting that experiments like ours are routinely performed upon evaluating and adopting a new DNN architecture, hence, obtaining results similar to those in Fig. 2.2(right) comes at a modest cost in terms of additional work. Moreover, transfer learning could be leveraged to further extend the applicability of the available experiments.

2.6 The RightTrain Solution

As proved in Sec. 2.7, directly optimizing (2.10) subject to constraints (2.11), is a daunting task. We thus introduce a new, effective heuristic, called RightTrain, which *decouples* the decisions of (i) choosing the instance tree, (ii) performing instance-to-node mapping, and (iii) assigning the necessary resources. At every step, *efficiency* is the main criterion driving RightTrain decisions.

As summarized in Fig. 2.3, RightTrain takes as an input the set of instance trees to consider (like the one in Fig. 2.1(b)); such a set can be efficiently computed offline, in a scenario- and application-dependent manner. RightTrain then iterates over the set of instance trees, selecting at each step the one requiring the least amount of *total* processing (Step 1 in Fig. 2.3, detailed in Sec. 2.6.1). For each tree, the y-variables are fixed, hence, in Step 2 (Sec. 2.6.2) we make the mapping decisions z, under the (temporary) assumption that (i) all the data of the selected sources is used, and (ii) all the processing capabilities at each node are exploited. Both assumptions

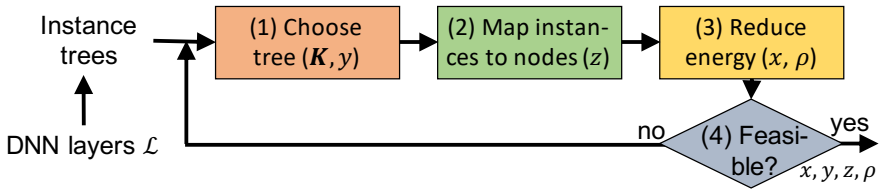


Fig. 2.3 Main steps of the RightTrain solution concept: given the set of layer instance trees to consider, RightTrain selects, at each iteration, the hitherto-untested tree associated with the lowest energy consumption (Step 1, Sec. 2.6.1). For such a tree, it makes the near-optimal layer instance-to-node mapping decisions (Step 2, Sec. 2.6.2), and further improves efficiency by tweaking data and resource utilization (Step 3, Sec. 2.6.3).

are dropped in Step 3 (Sec. 2.6.3), which seeks to refine the solution obtained in Step 2 by using less data and/or less computing power, thereby reducing the energy consumption without jeopardizing the learning performance. If a feasible solution is obtained, then the algorithm terminates (Step 4); otherwise, it goes back to Step 1 and moves to the next instance tree.

2.6.1 Layer instance tree ordering

Step 1 of the RightTrain solution requires choosing, from the set of layer instance trees to consider, the next one to try. Ideally, we would like to select a tree minimizing the energy consumption (2.10), however, this is not possible as instance-to-node mapping and resource assignment decisions (respectively, Steps 2 and 3) have yet to be made. Nonetheless, the instance tree – along with information on layer and data source characteristics – allows for estimating the total quantity of *processing* entailed by the tree itself. Specifically, recalling that $\mathcal{I}$ is the set of layer instances created for a given tree (i.e., DNN structure) and that y-decisions represent the tree topology, we can express the amount of required processing as:

$$\mathbf{K}(y, \epsilon^{\max}) \sum_{l:\,(l,i)\in\mathcal{I}} r(l) \sum_{\substack{d\in\mathcal{D}:\,(l,i)\text{ is}\\ \text{an ancestor of } d}} \Delta(d) \prod_{\substack{m\in\mathcal{L}\text{ in path}\\ \text{from } d \text{ to } l}} q(m). \tag{2.20}$$

Looking at (2.20) from right to left, the processing required by a given layer of a DNN for each epoch depends upon [27]: (i) the quantity of data it processes (which in turn depends upon the q-coefficients of the layers traversed before l) and (ii) the layer complexity. Such a quantity is then summed across all layer instances, and multiplied by the number $\mathbf{K}(y, \epsilon^{\max})$ of epochs to run before convergence.

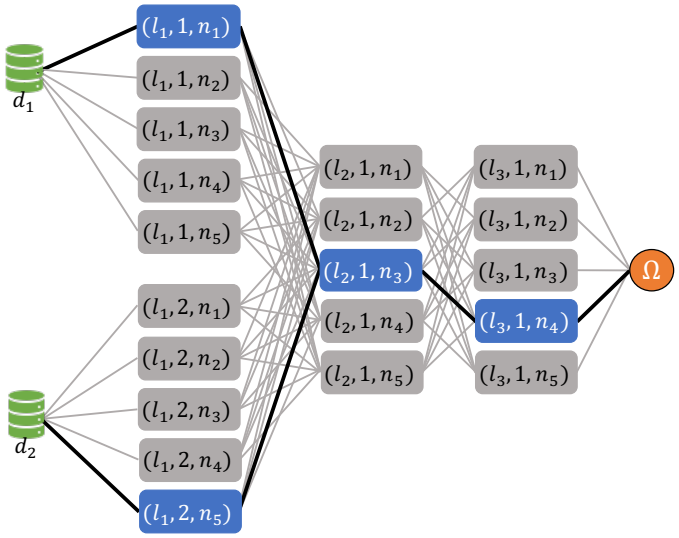


Fig. 2.4 The expanded graph representing the possible decisions in Fig. 2.1, with colored nodes and bold edges highlighting the DA-ST corresponding to mapping decisions in Fig. 2.1(c).

In addition to being a sound criterion to follow in all cases, selecting layer instance trees associated with low processing load (2.20) often results, as proved in Sec. 2.7, in selecting trees yielding a low value of the objective (2.10). Indeed, energy consumption (2.19) is often dominated by the processing energy (2.17), which in turn depends upon three quantities also accounted for in (2.20), namely, the number $\mathbf{K}$ of iterations, the layer complexity $r(l)$, and the quantity of available data Δ.

2.6.2 Layer instance-to-node mapping

Step 2 of RightTrain *maps* layer instances in $\mathcal{I}$ to nodes in $\mathcal{N}$, i.e., it decides at which node each layer instance should run. As mentioned above, initially such decisions are made under the assumption that all available data and computational capabilities are used. This problem is combinatorial and, in general, hard to approach. On the positive side, however, we can leverage the *tree* structure connecting layer instances, as in Fig. 2.1(b). Specifically, our approach is to (i) build the *expanded graph* shown in Fig. 2.4, summarizing all possible mapping decisions, and (ii) build a delay-aware Steiner tree (DA-ST) upon such a graph. The DA-ST is the minimum-weight tree spanning a given subset of the vertices of an undirected graph, called *terminal*

Algorithm 1 Layer instance-to-node mapping

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

vertices, with the additional constraint that the maximum learning time $T^{\max}$ must be honored, as per the $T^{\max}$-clause in (2.11).

The expanded graph is built as follows:

1. it contains a vertex (l,i,n) for each possible deployment decision, i.e., for each layer instance-node pair such that $\mu(l,i,n) = 1$;

2. an edge is drawn between vertices (l,i,n) and (m,j,n') if the layer instances are connected in the instance graph and the nodes n and n' can communicate, i.e., if $y(l,i,m,j) = 1 \ \wedge \ S(n,n') > 0$;

3. we also create an additional vertex for each data source in $d \in \mathcal{D}$, and connect such a vertex to all vertices in the expanded graph representing layer instances to which d is connected;

4. we add a further vertex Ω connected to all vertices (l_L,i,n) corresponding to the last layer l_L. Importantly, the vertex Ω allows us to build trees without restrictions on the number of instances of the last layer l_L;

5. the weight of each edge $((l,i,n),(m,j,n'))$ corresponds to the energy consumption, as per (2.17), due to the deployment decision represented by vertex (m,j,n'), i.e., running instance (m,j) at node n'. Notably, we do not consider the energy consumed for data transmission defined in (2.18), as it is assumed negligible compared to the computational energy.

As exemplified in Fig. 2.4, the DA-ST connects the vertices representing the data sources with Ω, and covers the same topology as the layer instance tree. Since the DA-ST is the *minimum-weight* among the trees with these features, its vertices represent the layer instance-to-node mapping that minimizes energy consumption (see (2.10)), under the aforementioned conditions, i.e., that all data and computation resources are used. The mapping decisions are made as summarized in Alg. 1. Given the expanded graph whose vertices are (i) data sources in $\mathcal{D}$, (ii) vertices of type (l, i, n) representing possible mappings, and (iii) special vertex Ω (Line 0), we build the DA-ST $\mathcal{T}$. In Line 1, we initialize the tree $\mathcal{T}$, so as to include vertex Ω. Then, so long as there are data sources not yet included in the tree (Line 2), we look for the minimum-weight path such that (i) it connects a data source d not yet in the tree with a vertex v of the DA-ST itself, and (ii) does not break constraints (2.11). To this end, in Line 3 the minimum weight $w^\star$ and the minimum-weight path $\pi^\star$ are initialized to ∞ and $\emptyset$, respectively. Then function $\mathsf{RestrictedMinWeightPath}(d, v)$ (Line 5) provides the path π connecting each data source d not yet reached by the tree with each vertex v already in the tree. The minimum-weight path is then identified (Line 7) and added to the DA-ST $\mathcal{T}$ in Line 8. Once all data sources have been included, the DA-ST is completed. The algorithm therefore sets to 1 the z-variable corresponding to the selected DA-ST vertices (Line 10), and returns them.

The structure of Alg. 1 mimics the algorithm in [63], which solves approximately the Steiner tree problem. The procedure $\mathsf{RestrictedMinWeightPath}$ uses the algorithm proposed in [64] to find the minimum-weight path connecting v and d while honoring delay requirements. Finding such a path requires solving an instance of the *constrained shortest path* problem. The problem itself is NP-hard, however, the heuristic [64] can solve it within ϵ ($\epsilon \geq 0$) from the optimum in polynomial time.

As proven in Sec. 2.7, Alg. 1 as a whole has polynomial (namely, $O\left(|\mathcal{D}||\mathcal{I}|^3|\mathcal{N}|^3\frac{1}{\epsilon}\right)$ time complexity; also, if the algorithm finds a feasible solution, it can be proved that it is within $\left(2 - \frac{2}{W}\right)(1+\epsilon)$ from the optimum, where W is the number of vertices of the optimal DA-ST. Importantly, the feasibility of the final solution, i.e., after the refinement, is checked in Step 4.

2.6.3 Decisions refinement

In Step 1 and 2, we have decided which layer instances to create and how to connect them, i.e., the layer instances in $\mathcal{I}$ and the edges in the layer instance tree connecting them, as expressed through the y-variables, and how to map layer instances to physical nodes, i.e., the z-variables. The values of both variable sets have been obtained under the assumption that all data from the selected data sources and all the capabilities of physical nodes are used. In the spirit of recent efficiency-focused research [27], Step 3 seeks to establish whether *all* that data and that computational power is really needed. Our goal is thus to obtain a solution that meets all constraints in (2.11), including the minimum learning quality and maximum time, while further improving the energy objective (2.10).

Given the y- and z-variables, the problem of optimizing (2.10) subject to constraints (2.11) only has continuous variables, namely, x and ρ. It follows that such a problem can be efficiently solved: (i) by off-the-shelf solvers, e.g., CPLEX or Gurobi, if a closed-form expression is available for $\mathbf{K}$, $\mathbf{E}$ and $\mathbf{T}$, or (ii) through iterative, gradient-based methods like BFGS [65], if such closed-form expressions are not available. Even more importantly, as formally proven in Sec. 2.7, such a continuous problem is *convex* in many practical cases. It follows that numerical approaches (either off-the-shelf solvers or gradient-based methods) are *guaranteed* to find the optimal values of x and ρ with very good efficiency – polynomial worst-case complexity [66], and much faster than that in most cases.

Finally, after the decisions refinement, Step 4 checks for the feasibility of the found solution; if the latter is not feasible, RightTrain goes back to Step 1, selects a new tree and repeats the process until a feasible solution is found.

2.7 Problem and Algorithm Analysis

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

Property 1 (Problem hardness)**.** *The problem of optimizing (2.10) subject to constraints (2.11) is NP-hard.*

Proof. We prove the thesis through a reduction from the Generalized Assignment Problem [67] (GAP), requiring to assign a set of *tasks* to a set of *agents*; each task-to-agent assignment incurs a given *cost*, and the goal is to minimize the total cost. More specifically, we reduce instances of the GAP to *simpler* instances of our own problem, where:

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

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

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

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

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

We remark that the proof of Property 1 reduces GAP instances to greatly simplified instances of our own problem; this allows us to conjecture that our

problem, besides being NP-hard, is also significantly more complex than an NP-hard problem like GAP.

Let us now move to the RightTrain heuristic, and focus on Step 1 therein, i.e., the choice of the next layer instance graph to consider. In Sec. 2.6.1, we formulated a selection criterion based on (2.20), expressing the quantity of processing associated with a given tree. Next, we prove that such criterion often results in selecting the instance tree with the lowest energy consumption.

Property 2 (Processing and energy)**.** *If proportional energy factors are the same for all usable nodes and dominate the global energy consumption, then a layer instance tree minimizing the quantity of processing (2.20) also minimizes the objective (2.10).*

Proof. The objective (2.10) requires minimizing energy, which is given in (2.19). Neglecting $e_{\text{net}}(n, n')$ and $e_{\text{f}}(n, l)$, then (2.19) reduces to $\sum_{(l,i)\in\mathcal{I}} t^{l,i}_{\text{comp}} e_{\text{p}}(\nu(l,i))\rho(l,i)$ which, recalling (2.12) and considering the conditions stated in Sec. 2.6.1 and that we are not making any mapping decision, the expression becomes

$$\sum_{(l,i)\in\mathcal{I}} e_{\text{p}}(\nu(l,i))r(l) \sum_{\substack{d\in\mathcal{D}\colon (l,i) \text{ is} \\ \text{an ancestor of } d}} \Delta(d). \tag{2.21}$$

Considering $e_{\text{p}}(n) = e_{\text{p}}\, \forall n$, we can re-write (2.10) as:

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

which is exactly $\frac{e_{\text{p}}}{\mathbf{K}(y,\epsilon^{\max})}$ times the quantity in (2.20). □

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

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

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

Proof. As shown in [64], the algorithm implementing the RestrictedMinWeightPath procedure has $O\left(mn(\log\log n\left(1+\frac{1}{\epsilon}\right)\right)$ time complexity, where m and n are, respectively, the number of vertices and edges of the input graph. In our case, the

number of vertices of the expanded graph is $O(|\mathcal{I}||\mathcal{N}|)$, and the number of its edges is $O(|\mathcal{I}|^2|\mathcal{N}|^2)$. The whole procedure is repeated at most $|\mathcal{D}|$ times, hence, the thesis follows. □

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

Proof. The structure of Alg. 1 mimics that of the algorithm in [63] to solve the Steiner tree problem, which has a competitive ratio of $2 - \frac{2}{W} \leq 2$, where W is the number of vertices in the optimal Steiner tree. However, Alg. 1 contains a further source of suboptimality, namely, the RestrictedMinWeightPath procedure; as per [64], its result is guaranteed to be within $(1+\epsilon)$ from the optimum. Combining the two competitive ratios, the thesis follows. □

As for Step 3 of the RightTrain approach, it requires setting the x and ρ variables so as to further improve the energy objective (2.10), without jeopardizing the constraints in (2.11). By leveraging theoretical arguments and experimental observations, we can state the following result.

Proposition 1 (Convexity of the problem in Step 3)**.** *The problem of optimizing (2.10) subject to constraints in (2.11), with the y- and z-values fixed, is convex.*

The arguments supporting Proposition 1 can be summarized as follows. Constraints (2.2)–(2.9) are clearly linear in the variables x and ρ, once y and z are given. Similarly, constraints (2.13)–(2.19) reduce to linear expressions in variables x and ρ. Also, (2.12) is convex in $\rho(l,i)$, as it is easy to verify that its second derivative is always positive. As for the number of iterations needed for convergence, although no closed-form expression for $\mathbf{K}$ is available, theoretical and experimental works [68–70] all concur that the relationship between the quantity of used data and the resulting learning quality (e.g., accuracy) is best captured by logarithmic functions, which are convex.

2.8 Performance Evaluation

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

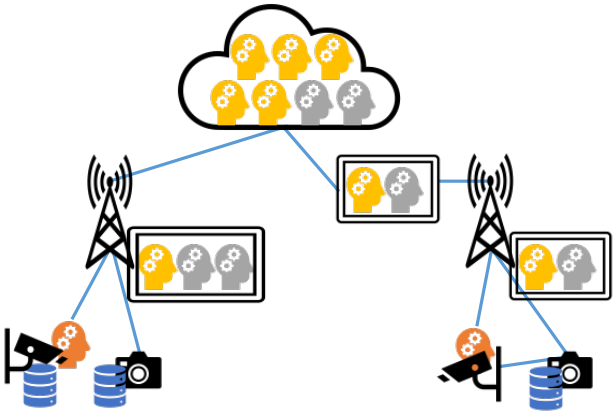


Fig. 2.5 An example three-layered scenario, including mobile, edge, and cloud nodes. Brains denote different devices with computational capability, with the color of the brain corresponding to the category of the device itself (gold, silver, or bronze). Dark-blue cylinders denote data sources; light-blue edges connect pairs of devices that can communicate.

Table 2.1 Complexity of the layers of the AlexNet DNN used for our performance evaluation

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

2.8.1 Reference scenarios

Learning task and DNN. We consider an image classification task over the CIFAR dataset, using a version of the AlexNet DNN [2], including five convolutional layers and three fully-connected ones. Tab. 2.1 summarizes the layers composing the AlexNet DNN, along with their complexity (per sample), expressed in millions of operations (MOPs) per sample. Both CIFAR and AlexNet are well-known, widely available and well-studied; this makes our results more significant and easier to reproduce and generalize.

Table 2.2 Computational capability and power consumption of gold, silver, and bronze servers

Class	Real-world example	Capability [TOPS]	Power consumption [W]	Efficiency [W/TOPs]
Gold	NVIDIA A100 [71]	312	400	1.28
Silver	NVIDIA A4000 [72]	153.4	140	0.91
Bronze	Apple A14 bionic [73]	11	6	0.54

Network scenarios. We consider three-tier scenarios like the one exemplified in Fig. 2.5, featuring the mobile-edge-cloud continuum and including:

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

The computational capabilities of UEs are rated *bronze*, i.e., very limited. Edge- and cloud-level servers, instead, come in *silver* and *gold* variants, the latter with very large, computational capabilities. Importantly, as can be seen from Tab. 2.2, lower-capability servers have better efficiency, i.e., need fewer watts to provide the same number of trillions of operations (TOPs). This suggests that being able to exploit *all* elements of the mobile-edge-cloud continuum, including using less-powerful devices for moderate loads, is an important asset for any decision-making strategy.

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

Benchmark strategies. We compare the performance of RightTrain against SL [24], owing to its power and performance [26]. Specifically, SL splits the DNN into three parts, and aims at running one at each of the mobile, edge, and cloud layers of the network topology; for each layer, the viable server resulting in the lowest energy consumption is chosen. Since we are interested in the *best* decisions that can be made under the SL paradigm, we compare all possible splits, and choose the one resulting in the best value of the objective (2.10).

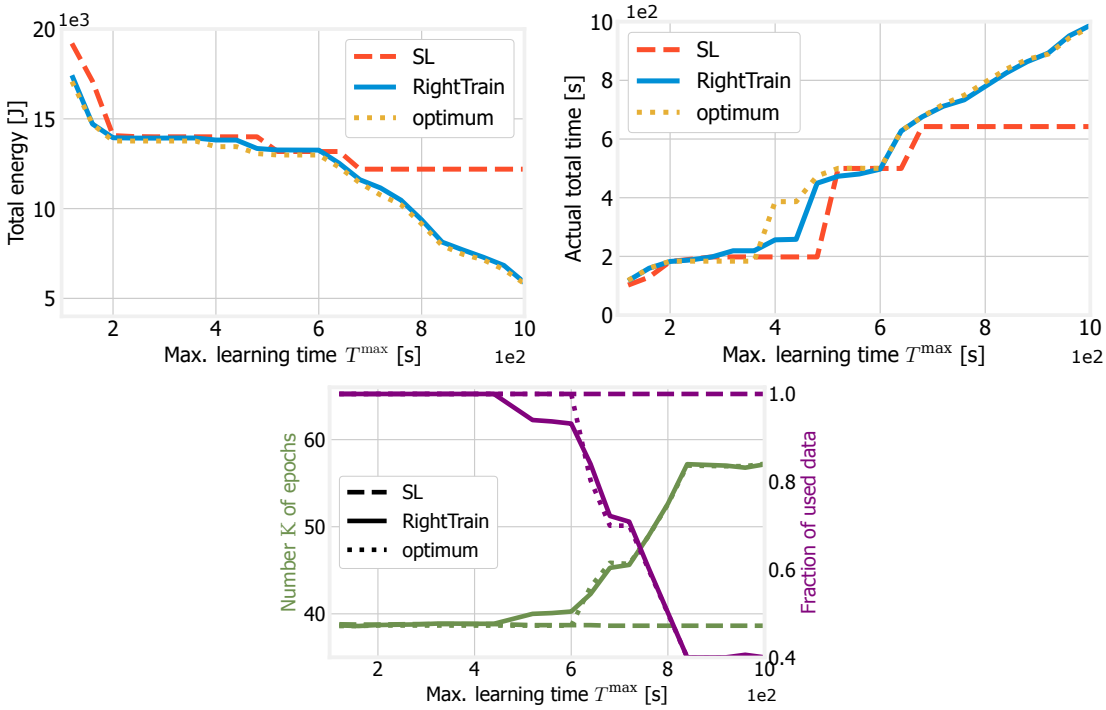


Fig. 2.6 Small-scale scenario: energy consumed as a function of the maximum learning time $T^{\max}$ (left), actual and maximum learning time (center), number of iterations and fraction of used data (right).

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

2.8.2 Numerical results

The most basic aspect in which we are interested is how effective RightTrain and its counterparts are in pursuing the optimization objective (2.10). To this end, Fig. 2.6(left) shows the energy consumed as a function of the maximum learning time $T^{\max}$, for the small-scale scenario. Consistently with intuition, lower values of $T^{\max}$, hence, tighter delay constraints, result in a higher energy consumption.

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

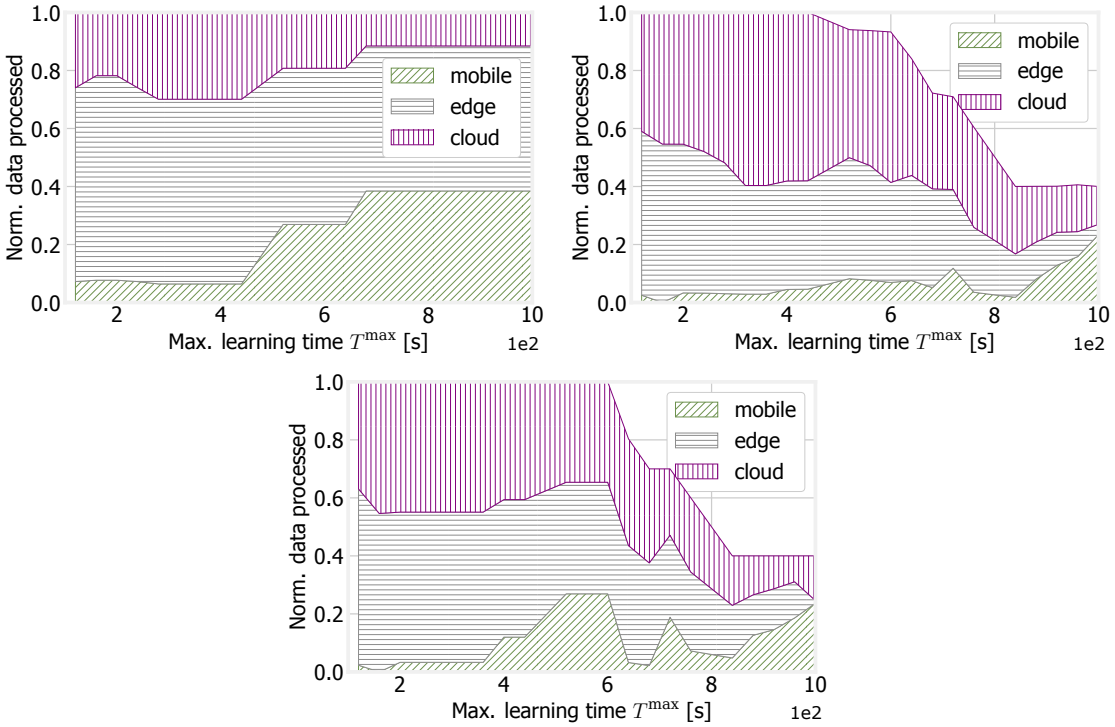


Fig. 2.7 Small-scale scenario: quantity of data processed at different parts of the network topology as the maximum learning time $T^{\max}$ varies, under the SL (left), RightTrain (center), and optimal (right) strategies.

with SL stops decreasing, while RightTrain is able to track the optimum and yield a substantially lower energy consumption, over 50% less than SL. The reason for such behavior is shown in Fig. 2.6(center): SL can result in learning times that are shorter than $T^{\max}$, especially when $T^{\max}$ itself is higher.

One reason for this is shown in Fig. 2.6(right), portraying the fraction of data used by each strategy (purple) and the resulting number of iterations K (green). We can see that SL (dashed lines) always uses all available data, which results in a constant (and low) number of iterations. On the other hand, both RightTrain and the optimum are able to use less data when the delay restrictions are looser, achieving a lower energy consumption and, hence, a better efficiency, in spite of a higher number of iterations.

The second reason is shown in Fig. 2.7, depicting how each strategy utilizes the different parts of the network topology. We can observe that SL (left plot) tends to use more low-powered mobile nodes as $T^{\max}$ increases, as one might expect. For RightTrain (center plot) and the optimum (right plot), the quantity of data to

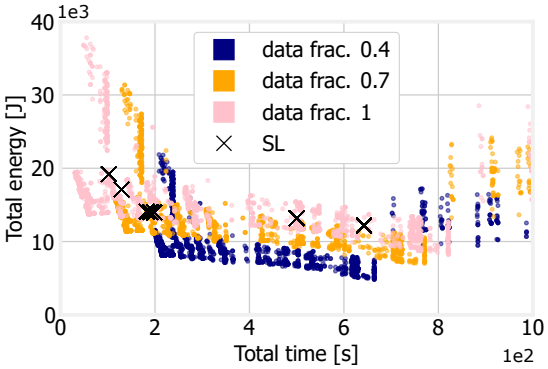


Fig. 2.8 Small-scale scenario: energy/time trade-offs possible under the RightTrain (dots) and SL (crosses) strategies. Dots of different colors correspond to different fractions of used data.

process decreases as $T^{\max}$ increases, which allows for a greater flexibility in using all segments of the mobile-edge-cloud continuum, including high-powered cloud nodes when appropriate. Notice that, under RightTrain and (to an even greater extent) the optimal strategy, the curves in Fig. 2.7 do not look smooth, e.g., the quantity of data processed at mobile nodes fluctuates as $T^{\max}$ increases. This is in contrast with the monotonic evolution in Fig. 2.7(left), and reflects the fact that RightTrain is better than SL at accounting for the nonlinearities of the system behavior (e.g., the fixed energy component e_{f}) and it adjusts its decisions accordingly.

Fig. 2.8 provides further insights about the greater flexibility of RightTrain compared to SL. Each marker in the plot corresponds to a possible solution, with its position along the x- and y-axes corresponding, respectively, to its learning time and energy. Dots represent solutions reachable by RightTrain, with their color corresponding to the fraction of used data; black crosses represent solutions reachable by SL. We can immediately see that being able to not use all data allows RightTrain to explore a larger set of high-quality trade-offs, often with a smaller energy consumption and longer learning time. As for SL, all of the solutions it can explore can also be reached by RightTrain when all data is used (pink dots).

We now move to the large-scale scenario and plot, in Fig. 2.9(left), the energy consumed by the SL and RightTrain strategies (indeed, owing to the scenario size, computing the optimum is not feasible). It is possible to observe a similar behavior to that in Fig. 2.6(left), with RightTrain always yielding a smaller power consumption than SL, and the difference growing as $T^{\max}$ gets larger. By comparing Fig. 2.9(left) to Fig. 2.6(left), it is also possible to observe how RightTrain performs noticeably

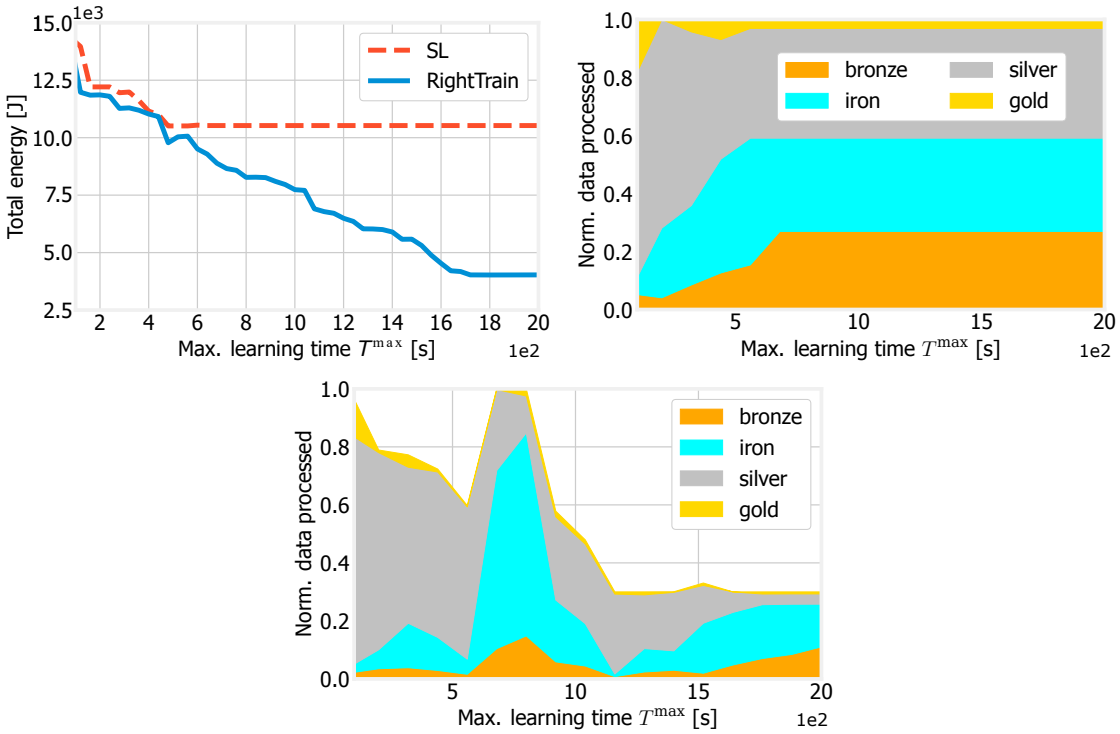


Fig. 2.9 Large-scale scenario: energy consumed as a function of the maximum learning time $T^{\max}$ (left), and quantity of data processed at nodes of different classes under the SL (center) and RightTrain (right) strategies.

better than SL even for small values of $T^{\max}$, and how the two curves diverge earlier in Fig. 2.9(left) than in Fig. 2.6(left).

The reason for such a different behavior is presented in Fig. 2.9(center) and Fig. 2.9(right), depicting how (respectively) SL and RightTrain use the different types of nodes in the topology. Similarly to Fig. 2.7, RightTrain can more flexibly – one would almost say, *creatively* – use available physical nodes, including "iron" ones, thus yielding a lower energy consumption than SL. The behavior and performance difference is more clear here than for the small-scale scenario, due to the wider variety of existing nodes.

2.9 Testbed validation

We now validate our model and approach through a lab test-bed composed of three nodes, depicted in Fig. 2.10:

Fig. 2.10 The nodes of the lab test-bed we employ.

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

Laptops run the Ubuntu 18.04 operating system, while the RPi runs Ubuntu Server 20.04. UE and edge node are connected through a 3GPP LTE virtualized Radio Access Network (vRAN), leveraging Ettus Universal Software Radio Peripheral (USRP) B210 boards. The vRAN is based on the srsRAN [74] open-source LTE stack implementation, which is compliant with LTE Release 9. The RPi is connected to the UE through Wi-Fi, with the latter acting as an access point.

As the learning activity to perform, we consider an image classification task over the CIFAR-10 [62] dataset using the Lenet DNN [75], composed of two convolutional layers followed by three linear ones. We study the performance and behavior of the possible (i.e., feasible) *mappings* between layers and physical nodes, under two test-bed configurations:

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

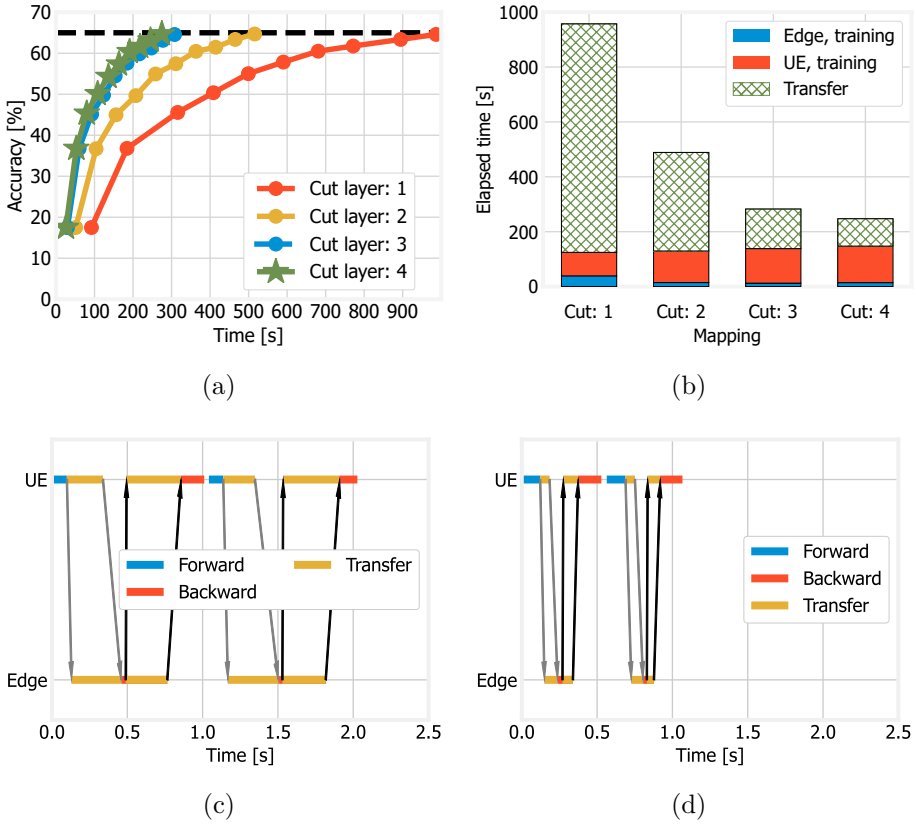


Fig. 2.11 Lab test-bed, two-node configuration: accuracy vs. time for different mappings (a); total elapsed time for different mappings (b); Gantt chart for the "cut layer: 2" (c) and "cut layer: 4" (d) mappings.

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

Fig. 2.11 reports the results for the two-node configuration. From Fig. 2.11(a), we can observe that 10 epochs are always sufficient to reach the target accuracy; however, the *time* needed to perform such epochs changes significantly; specifically, the later we "cut" the network, the shorter the learning time. The reason of this behavior is highlighted in Fig. 2.11(b). Interestingly, the total computing time (i.e., considering both the Edge node and the UE) remains roughly constant (since the Edge node and UE laptops have similar performance). On the contrary, the amount of data to be transmitted, hence, the data transfer time, decreases substantially (up to 73%) with higher values of the cut layer, i.e., if we cut the network after a larger number of layers. This is consistent with the fact that later layers of the DNN (i.e.,

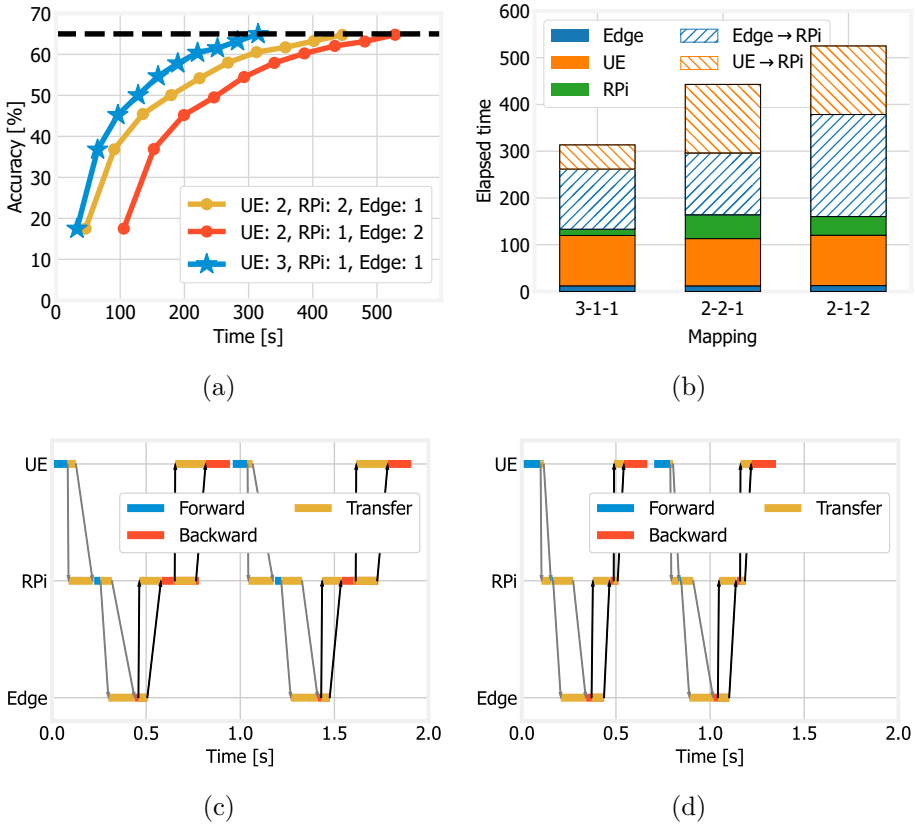


Fig. 2.12 Lab test-bed, three-node configuration: accuracy vs. time for different mappings (a); total elapsed time for different mappings (b); Gantt chart for the "UE: 3, RPi: 1, Edge node: 1" (c) and "UE: 2, RPi: 1, Edge node: 2" (d) mappings.

farther from input data) exchange less data, hence, "cutting" the DNN at such layers reduces the quantity of information to exchange between nodes.

Fig. 2.11(c) and Fig. 2.11(d) cast additional light on this phenomenon, by depicting how the two nodes alternate performing computations and exchanging data in the first two batches of a typical iteration. For each batch, each node performs the forward step and transmits the output of its own last layer to the following node. Then, when the Edge node terminates the forward stage of the last layer, it computes the loss and starts the backward procedure, computing the gradients and sending them back. The transmissions of the output and gradients are indicated respectively by the gray and black arrows in the plots. Blue and red bars therein correspond to forward and backward passes, with the latter always taking roughly twice as much as the former. This aligns with our expectations, as the computational cost of

the backward pass is approximately twice that of the forward due to the gradient computation. It is easy to notice that, while the forward and backward passes take roughly the same time, "cutting" the DNN at layer 2 (Fig. 2.11(c)) instead of layer 4 (Fig. 2.11(d)) results in a much larger quantity of data to transmit, hence, longer total training times.

Fig. 2.12 confirms the findings above, in spite of the fact that the DNN layers can be spread across three nodes, hence, mappings are more complex. As in Fig. 2.11(a), in Fig. 2.12(a) the accuracy reached at each epoch does not change, but the time such epochs take does depend upon the mapping. Importantly, such a time is deeply influenced, as shown in Fig. 2.12(b), by the transfer times between the nodes.

2.10 Conclusion

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

In the next chapter, we will address a similar problem, but with two key differences. First, we will assume a cooperative learning scenario in which each node trains an entire DNN, and not only a limited set of layers. Second, the training is performed sequentially by the learning nodes, or set thereof, with the model being pruned to accommodate nodes with limited computational capabilities. Thus, we will explore how DNNs' pruning can be utilized in Cooperative Learning and how to address the inherent challenges.

Chapter 3

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

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

Part of the work described in this chapter has been published in Malandrino, F., **Di Giacomo, G.**, Karamzade, A., Levorato, M., Chiasserini, C. F. (2023). Tuning

DNN Model Compression to Resource and Data Availability in Cooperative Training. *IEEE/ACM Transactions on Networking*, 32(2), 1600-1615.

3.1 Introduction

Training machine learning (ML) models is notoriously hard, as it requires large quantities of data as well as significant computational resources [76, 77]. To cope with this issue, cooperative training – most notably, federated learning (FL) [31, 77, 78] – has emerged as a nigh-universal approach to leverage the resources of multiple nodes to perform a single learning task. Examples range from smart factory scenarios [79], where model training takes place at both cloud- and edge-based servers, to space applications [80] where models are first trained on the ground and then refined aboard the spacecraft.

In all such scenarios, the data and resources needed to perform the training are scattered throughout different nodes, whose availability and connectivity may significantly vary in both space and time [81]. This results in a major technical challenge, namely, the *mutual adaptation* of the decisions concerning (i) ML training (e.g., model selection and compression), (ii) data selection (i.e., which nodes shall be asked to contribute their datasets for training), and (iii) node and resource allocation (i.e., at which network nodes to train each portion of an ML model). Importantly, the selection of datasets can be performed without sharing the data itself, but based solely on privacy-preserving statistical information [82, 83].

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

- i.i.d. datasets at different nodes – in this case, the main benefit is fine-tuning the resources committed for training to the required learning quality, e.g., by exploiting cheaper nodes;
- datasets that are not i.i.d. but related (e.g., from different domains [79]) – in this case, the same model can be successfully and efficiently trained to work with data from all domains;

- different datasets with convolutional DNNs – as convolutional layers recognize basic features of the data (e.g., simple shapes in images) as opposed to their meaning, their information can be transferred [84, 85] to models using different datasets.

As better discussed in Sec. 3.7, many existing works address one or another of the aforementioned aspects, but fall short of providing a comprehensive strategy to jointly make all the required decisions. To fill this gap, in this work we focus on deep neural networks (DNNs) and propose a solution strategy and algorithmic framework called Performance-Aware Compression and Training (PACT), supporting all three cases above. PACT creates optimized strategies for the training of DNNs, in presence of (a) heterogeneous nodes, whose datasets cannot be shared, and (b) different DNN models to choose from. A major novelty of PACT is the ability to leverage *multiple* DNN models across different stages of the same learning task, by *switching* among them as needed (e.g., through model pruning [86] or knowledge distillation [87]). For each stage – hence, for each model –, PACT then selects the most appropriate datasets, network nodes, and resources. As in the example depicted in Fig. 3.1, a fairly complex model may be used in the early stages of training, running on a small set of powerful nodes. Later, it is possible to switch to a simpler (e.g., pruned) model, thus including more nodes with smaller capabilities but more valuable local data [84]. At the same time, the benefits of model switching must be weighed against the cost of switching itself, which requires additional resources and will often result in a (temporary) drop in learning performance.

Our main contributions can be summarized as follows.

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

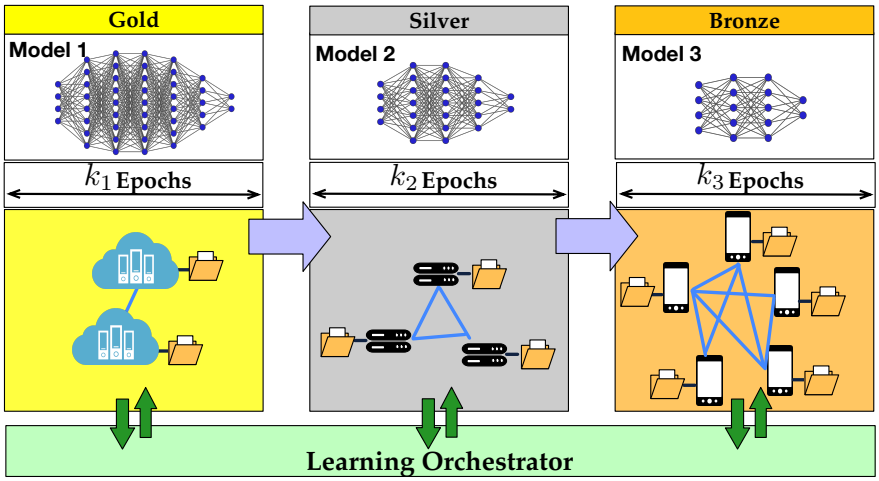


Fig. 3.1 Cooperative training process proposed and optimized in this chapter. Subsets of nodes sequentially train compressed versions of a DNN model. In the picture, nodes are categorized based on their computing capabilities and data availability, and, in the example, the training sequence is based on nodes' ranking (gold, silver, bronze). Our framework, named PACT and running at the learning orchestrator, optimizes the set of nodes, number of epochs, and model compression along the process.

existing works that only target one aspect at a time (e.g., choosing the mode) and consider the others immutable (e.g., resources are given).

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

(3) Performance evaluation: we evaluate the performance of PACT under three different real-world scenarios for distributed ML. In all cases, PACT yields training strategies that adapt to the existing resources and training data, honoring the target

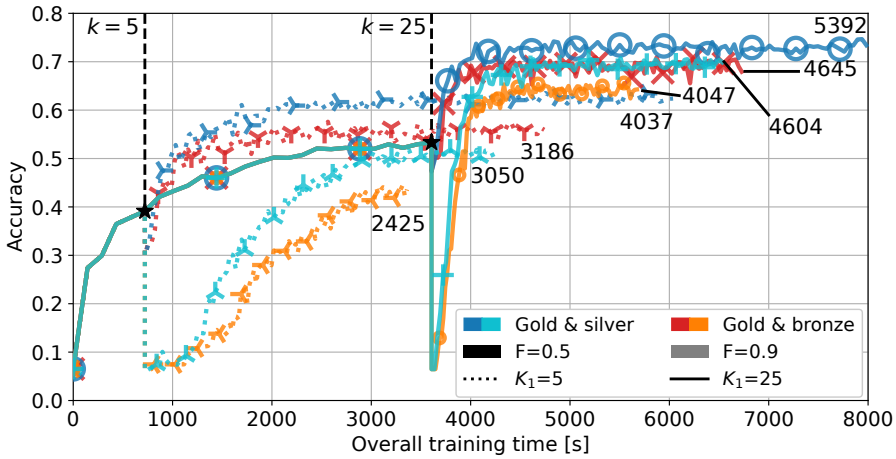


Fig. 3.2 Accuracy vs. training time for different pruning epoch K_1 (denoted by different line styles) and percentage F (denoted by different color shades). Upon pruning, a sudden drop in accuracy occurs. Cold and warm colors denote the set of nodes used for FL. Numbers indicate the total CPU time [s], while each marker corresponds to 10 epochs.

learning quality and time at a low energy cost. Specifically, PACT decisions are always very close (and, in many cases, identical) to the optimal ones, and substantially better than those made by state-of-the art approaches. We further show how PACT can recover from the effects of inaccurate estimations of the effect of model-switching decisions.

The rest of the chapter is organized as follows. Sec. 3.2 clarifies the problem we address, while Sec. 3.3 presents the system model and the decisions we tackle. Sec. 3.4 then introduces the methodology for estimating the loss as learning proceeds, and Sec. 3.5 describes our algorithmic solution. The obtained results are shown in Sec. 3.6; finally, Sec. 3.7 discusses relevant related work and Sec. 3.8 summarizes our conclusions.

3.2 A Motivating Example

In this section, we set in the first of the cases discussed in Sec. 3.1, i.e., i.i.d. datasets, and seek to illustrate the benefits of a cooperative training process that integrates model and nodes switching, but also emphasize the challenges in formulating and optimizing it. To this aim, we consider the case in which one of the most popular cooperative learning approaches, namely, federated learning (FL), is coupled with

model pruning [86]. The latter exploits the fact that, typically, many of a model's parameters have a small impact on its performance and can thus be pruned away, resulting in a DNN with similar performance but of lower complexity, and hence CPU and memory requirements. In particular, we evaluate the following scenario:

- the nodes perform an image classification task using the VGG-11 DNN model [88] as a starting point;
- FL uses the cross entropy loss function, batch size equal to 64, and the gradient descent optimizer with 10^{-3} learning rate and 0.9 momentum;
- the model is trained for K_1 epochs on 5 highly capable nodes ("gold" nodes), each using 8,000 randomly-chosen images from the CIFAR-10 dataset [62];
- then, a fraction F of the model's parameters is pruned[1];
- finally, training resumes adding 2 more learning nodes, with lower computing capability and fewer data: either "silver" with half the computing resources of the gold nodes and 2,500 local images each, or "bronze" with one third of the computing resources of the gold nodes and 750 local images each.

Three decisions should be made: (i) the number K_1 of epochs to execute before pruning, (ii) the percentage F of parameters to prune, and (iii) whether to use the "silver" or "bronze" nodes when resuming training. Notice how the first two decisions concern selecting and switching among models, while the third deals with the physical nodes participating in the learning process. Fig. 3.2 summarizes the effects of such decisions[2], which lead to the following main remarks.
Observation 1: Pruning more (i.e., F=0.9, orange and light blue curves) significantly reduces both CPU consumption (indicated by the numbers in the plot) and epoch duration (markers are closer to each other), thus speeding up the overall learning process and reducing its cost.
Observation 2: Larger values of K_1 (solid lines) are associated with better performance after pruning.
Observation 3: Using lower-capability ("bronze") nodes after pruning (warm colors) results in a larger difference between the learning performance obtained when K_1 is

[1]We use the PyTorch method [89] to set to 0 the weights with smaller L^2 norm, and the Simplify library [90] to remove them from the DNN.
[2]Only some values of F are possible, as we apply structured pruning.

small (i.e., 5) and when K_1 is larger (i.e., 25). Thus, achieving better performance while exploiting lower-capability nodes requires switching model later.

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

3.3 System Model and Problem Formulation

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

3.3.1 Model components

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

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

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

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

$$\Delta T(k)=\tau^{\text{change}}(m(k-1),n(k-1),m(k),n(k))+\tau^{\text{run}}(m(k),n(k)), \tag{3.1}$$

$$\Delta E(k)=\epsilon^{\text{change}}(m(k-1),n(k-1),m(k),n(k))+\epsilon^{\text{run}}(m(k),n(k)). \tag{3.2}$$

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

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

$$\Delta\ell(k)=\lambda^{\text{change}}(k,m(k-1),m(k))+\lambda^{\text{run}}(k,m(k),n(k)). \tag{3.3}$$

Again, (3.3) includes two components: λ^{change} – the contribution of transitioning from the previous to the current model (if a model switch is performed), and λ^{run} – the effect of training that model for an epoch. The sum of these components gives the difference between the loss at the current epoch k and that of epoch $k-1$, i.e., the result of the action enacted at k.

λ^{run} and λ^{change} describe two different actions with different outcomes: the former corresponds to the usual learning procedure, i.e., running one epoch of training; the latter corresponds to switching across different models, an operation that is done only occasionally and often results in a short-term degradation of the loss. Accordingly, the two components may have different signs: $\lambda^{\text{run}} \leq 0$ (the loss decreases) in most cases, while it is possible that $\lambda^{\text{change}} \geq 0$, as changing model may increase the loss value [91, 92]. Furthermore, the fact that λ^{run} also depends upon the used data allows our model to capture the familiar notion that some datasets are more useful for learning than others. Impossible transitions between learning settings are associated with $\lambda^{\text{change}} = \infty$.

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

Example 1 (System scenario)**.** *Consider the scenario in Sec. 3.2. In that case, the set of models $\mathcal{M}$ contains (i) the original model being used, plus (ii) one additional element for each possible pruning level*[3]*. For instance, there could be one element for "50% pruning" and another for "90% pruning". Thus, each value of fraction F to prune maps into a different element of $\mathcal{M}$. Concerning number K_1 of epochs to run before pruning, it corresponds to the epoch at which we change the model m being used, hence, $m(K_1) \neq m(K_1 + 1)$.*

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

In practice, the choice of the elements in $\mathcal{M}$, i.e., the *possible* models to consider, will be done based upon expert – possibly, domain- or scenario-specific – knowledge. A further aspect worth taking into consideration is the availability of information about models, e.g., whether the impact on the learning quality has been profiled as per Sec. 3.4 below.

3.3.2 Problem definition

Given the impelling need to make ML sustainable [93, 94], our goal is to minimize the overall learning energy consumption, while ensuring that the loss drops below a target value $\ell^{\max}$ within time $T^{\max}$. Importantly, and unlike many related works, it is not our objective to maximize the learning quality, but rather to minimize the energy consumption subject to learning quality and time targets.

Specifically, for each epoch k, the learning orchestrator has to select (i) which model $m(k)$ to train in epoch k, and (ii) which set $n(k)$ of nodes to involve next in the learning process. Based on these decisions, the values $\Delta T(k)$, $\Delta E(k)$, and $\Delta \ell(k)$ follow, expressing, respectively, how long iteration k takes, how much energy it consumes, and what improvement in the learning it yields. The learning orchestrator is assumed

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

to own a synthetic dataset, which can be either sampled from the participating devices, or obtained through an already trained generative model [95]. By exploiting the testing dataset, it is possible to assess not only the training loss, but also the test loss; using the latter results in better decisions and a lower risk of overfitting. The learning orchestrator acts based on the knowledge of the characteristics of the network nodes that can contribute to a learning process, and of the computational, temporal, and energy impact of running a model. Such values can indeed be calculated following, e.g., the methodology in [96].

The reason why methodologies like [96] are effective is that, contrary to intuition, the operations required by performing one training epoch of a DNN are *deterministic*, e.g., a certain number of matrix products and inversions. Accordingly, given the DNN model to train and the architecture/capabilities of the nodes employ, both the time and energy consumed can be known with virtual certainty. Thus, sets $\mathcal{M}$ and $\mathcal{N}$, as well as functions τ^{run}, τ^{change}, ϵ^{run} and ϵ^{change}, are given from the viewpoint of our problem.

On the contrary, λ^{run} and λ^{change} can only be estimated by the learning orchestrator, through estimators $\hat{\lambda}^{\text{run}}$ and $\hat{\lambda}^{\text{change}}$. This reflects the fact that understanding how training a specific model over specific nodes (hence, also data) improves learning is a hard problem, and, indeed, all existing works merely provide approximations and/or bounds to such quantities. In the following, we treat those estimators as given; then, in Sec. 3.4 we demonstrate one possible methodology that the learning orchestrator can use to compute them.

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

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

3.3.3 ADP formulation

First, we define the state space, set of actions, and cost function. The state at epoch k is given by $\mathbf{s}(k)=(k,\ell(k),T(k),m,n)$, while the set of actions available from state $\mathbf{s}(k)$ is given by all possible decisions $(m',n')\in\mathcal{M}\times\mathcal{N}$ such that the switch they entail (if any) is feasible. The *cost function* $\mathbf{C}(\mathbf{s}(k),\mathbf{a}(k))$ expresses the (immediate) cost of executing action $\mathbf{a}$ while in state $\mathbf{s}$ at epoch k, as the corresponding consumed energy $\mathbf{C}(\mathbf{s}(k),\mathbf{a}(k))=\Delta E(k)$. Such a cost comes directly from (3.2), i.e., $\mathbf{C}(\mathbf{s}(k),\mathbf{a}(k))=\epsilon^{\text{change}}(m,n,m',n')+\epsilon^{\text{run}}(m',n')$.

The *value function* $\mathbf{V}(\mathbf{s}(k))$, i.e., how desirable it is to be in state $\mathbf{s}(k)$, requires a more sophisticated, and domain-specific, definition. We set the value of being in state $\mathbf{s}(k)$ equal to 0 when, after $T^{\max}$, the loss is above $\ell^{\max}$; we set it to the maximum value (i.e., 1) whenever $\ell(k)<\ell^{\max}$ while $T(k)\leq T^{\max}$. For all other states, we compare the current loss $\ell(k)$ and time $T(k)$ with an *ideal* loss-time curve $\ell^{\text{ideal}}(t)$ which: (i) starts at $\ell(0)$ for T=0; (ii) ends at $\ell^{\max}$ for T=$T^{\max}$, and (iii) follows a power law in the between. The latter comes from the finding invariably reported in both theoretical [97–99] and experimental [100] works. Then, we can write the value of being in state $\mathbf{s}(k)$ as the difference between ideal and real loss values, i.e.,

$$\mathbf{V}(\mathbf{s}(k))=\mathsf{logistic}\left(\ell^{\text{ideal}}(T(k))-\ell(k)\right), \tag{3.4}$$

where the value is normalized via a logistic function.

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

$$\min_{\mathbf{a}(k)\in\mathbf{A}^k}\sum_k \mathbf{C}(\mathbf{s}(k),\mathbf{a}(k)) \tag{3.5}$$

$$\text{s.t. } \mathbf{V}(\mathbf{s}(K))=1 \;;\; T(K)\leq T^{\max}. \tag{3.6}$$

To solve our problem in real-world scenarios, however, there are two major challenges to face. First, the learning orchestrator does not have access to the future decrease (or increase) in the loss value $\Delta\ell(k)$, and how our decisions influence it. A possible solution to this issue is to use traditional Deep Reinforcement Learning (DRL)

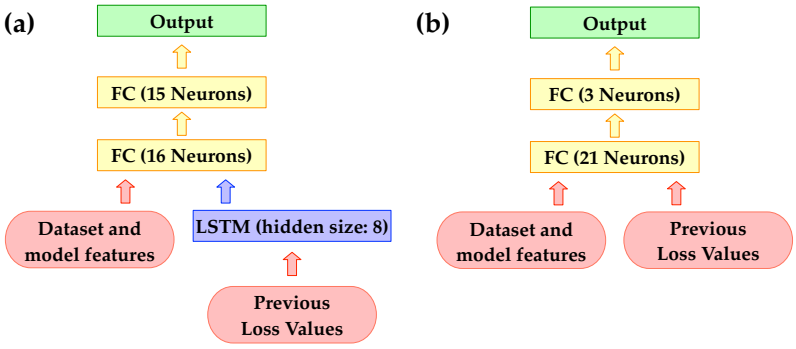


Fig. 3.3 Architecture of the $\hat{\lambda}^{\text{run}}$ (a) and $\hat{\lambda}^{\text{change}}$ (b) estimators.

approaches. For instance, Deep Q-Learning algorithms would implicitly learn the probabilistic dynamics of loss as a function of taken actions. However, training DRL agents often requires very large datasets to achieve satisfactory convergence, and may result in weak generalization. Herein, we take a different approach, where we build an ADP framework based on low-complexity neural networks (NN) estimators of possible loss trajectories with a finite time horizon. Second, in view of the number of possible actions, the learning orchestrator has to identify a subset of actions to evaluate at each epoch. Such challenges are dealt with in Sec. 3.4 and Sec. 3.5, respectively.

3.4 Estimating the Performance of Learning

As discussed in Sec. 3.3.2, neither of the quantities contributing to the loss evolution ($\lambda^{\text{change}}(k,m,m')$ and $\lambda^{\text{run}}(k,m,n)$) is known exactly. We thus introduce *estimators* for $\Delta\ell(k)$. Specifically, for $\lambda^{\text{run}}(k,m,n)$:

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

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

To obtain both the expected-value and the robust estimator, the learning orchestrator leverages the knowledge of the number of classes of the datasets owned by the nodes and makes use of NN architectures that can predict the expected testing loss variation as well as determine the prediction uncertainty. An approach we envision in the following is to estimate $\lambda^{\text{run}}(k, m, n)$ by leveraging the Long Short-Term Memory (LSTM) model in [101] and develop a similar, yet simpler, branched architecture, as depicted in Fig. 3.3(a). Importantly, alternative (possibly, more complex and/or comprehensive) approaches can be equally integrated with PACT.

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

As for $\lambda^{\text{change}}(k, m, m')$, since the goal is to predict the loss variation when we move from one DNN model to another, we leverage regression, using the NN in Fig. 3.3(b). The NN is fed the pruning ratio and the 5 loss values preceding the model switch. The regression model predicts the expected value $\hat{\lambda}^{\text{change}}_{\text{exp}}(k, m, m')$ as well as the 0.05 and 0.95 quantiles in the next epoch of the DNN training, with $\hat{\lambda}^{\text{change}}_{\text{rob}}(k, m, m')$ being again the 0.95 quantile.

We demonstrate our loss prediction in a small-scale example, as summarized in Fig. 3.4. We seek to model the testing loss attained by the AlexNet DNN over the CIFAR-10 dataset. The training happens in three steps:

1. the full DNN is trained for 15 epochs at a node containing 16,500 samples of classes 1–6 and 1,000 of classes 7–10;

2. the model is pruned with fraction F_1=0.25, and handed over to a new node owing 12,500 samples (representing all classes equally except 9–10, which are underrepresented) for 25 more training epochs;

3. the model is pruned with F_2=0.5 and handed over to a third node, owing 7,500 uniformly-distributed samples.

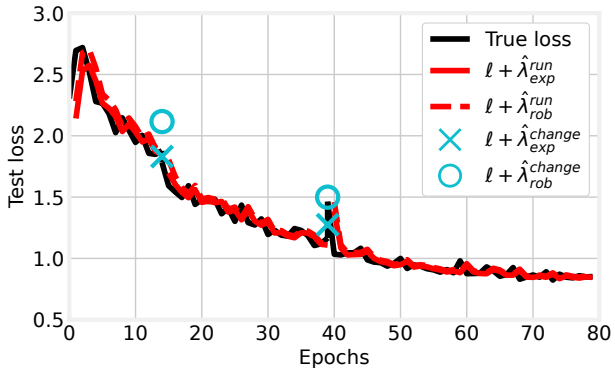


Fig. 3.4 Example of true loss vs expected-value and robust estimators.

In Fig. 3.4, the true loss is represented by the black line, while the red line and the blue markers represent, respectively, the predicted losses $\hat{\lambda}^{\text{run}}$ and $\hat{\lambda}^{\text{change}}$. It is possible to notice how, even in this relatively small-scale example, the estimators provide remarkably accurate predictions.

Finally, to improve the reliability of the robust estimator, the learning orchestrator compares the values obtained through the above NN to the lower bounds that are available for $\lambda^{\text{run}}(k, m, n)$ [35, Theorem 1] and for $\lambda^{\text{change}}(k, m, m')$ [91, Sec. 3]. If they result to be lower than the bounds, the latter are taken as robust estimators.

3.5 The PACT Algorithm

The goal of PACT is to let the learning orchestrator efficiently find high-quality solutions to the problem in (3.5), which, as shown later, is NP-hard. PACT consists of three steps:

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

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

- The *vertices* represent the states of the system; they are labeled with the current epoch k, model $m(k)$ and set of nodes $n(k)$ being used, and the total elapsed time $T(k)$ and current loss $\ell(k)$. With the aim of identifying feasible solutions, the latter quantity is computed using the robust estimators $\hat{\lambda}_{\text{rob}}^{\text{change}}(k, m, m')$ and $\hat{\lambda}_{\text{rob}}^{\text{run}}(k, m, n)$;
- Elapsed time and loss values are represented, respectively, with resolutions γ_T and γ_ℓ (e.g., if γ_ℓ=0.1, a vertex with ℓ=0.1 or 0.2 can exist, but not with ℓ=0.15);
- A directed *edge* is drawn between two vertices if there is an action making the system move from one corresponding state to the other; each edge is labeled with the *energy consumption* of the associated action, as in (3.2);
- Each vertex representing a feasible state of the system (i.e., with $\ell(k) \leq \ell^{\max}$ and $T(k) \leq T^{\max}$) is further connected to a virtual node Ω through a zero-cost edge.

The graph is created through the CreateExpandedGraph function, presented in Alg. 2. First, all *vertices* are created, representing all valid combinations of model and set of nodes, epoch, loss value, and elapsed time (Line 3–Line 6). Note that the quantization parameters γ_ℓ and γ_T (Line 4–Line 6) allow us to control the trade-off between size of the graph and quantization error.

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

Fig. 3.5 presents an example of expanded graph. The initial vertex is associated with epoch k=0, model $m(0)$=m_0, node $n(0)$=n_0, loss $\ell(0)$=1 and elapsed time $T(0)$=0. The learning target is $\ell^{\max}$=0.25 and the time limit is $T^{\max}$=1.5.

Algorithm 2 Creating the expanded graph

function CreateExpandedGraph
  $\mathcal{V} \leftarrow \{\Omega\}$ ▷ *set of vertices*
  for all $m \in \mathcal{M}, n \in \mathcal{N}$ **do**
    for all $k \in \left[1, 2, \ldots, \left\lceil \frac{T^{\max}}{\gamma_T} \right\rceil\right]$ **do**
      for all $\ell \in [0, \gamma_\ell, 2\gamma_\ell, \ldots, \ell(0)]$ **do**
        for all $T \in [0, \gamma_T, 2\gamma_T, \ldots, T^{\max}]$ **do**
          $v \leftarrow (k, m, n, \ell, T)$
          $\mathcal{V} \leftarrow \mathcal{V} \cup \{v\}$
  $\mathcal{E} \leftarrow \emptyset$ ▷ *set of edges*
  for all $v{=}(k, m, n, \ell, T){\in}\mathcal{V}$ **do**
    for all $\mathbf{a}{=}(m', n') \in \mathbf{A}$ **do**
      $T' \leftarrow T{+}\tau^{\text{change}}(m, n, m', n'){+}\tau^{\text{run}}(m, n)$
      $E \leftarrow \epsilon^{\text{change}}(m, n, m', n'){+}\epsilon^{\text{run}}(m', n')$
      if $T'{>}T^{\max} \vee E{=}\infty$ **then**
        continue ▷ *infeasible, skip this action*
      $\ell' \leftarrow \ell{+}\hat{\lambda}^{\text{change}}_{\text{rob}}(k, m, m'){+}\hat{\lambda}^{\text{run}}_{\text{rob}}(k, m, n')$
      $\ell' \leftarrow \gamma_\ell \left\lceil \frac{\ell'}{\gamma_\ell} \right\rceil$
      $T' \leftarrow \gamma_T \left\lceil \frac{T'}{\gamma_T} \right\rceil$
      $v' \leftarrow (k{+}1, m', n', \ell', T')$
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(v, v', \mathsf{weight}{=}E)\}$
    if $\ell \leq \ell^{\max} \wedge T{\leq}T^{\max}$ **then**
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{v, \Omega\}$ ▷ *feasible state*
  return $\mathcal{G}{=}(\mathcal{V}, \mathcal{E})$

Also, the resolution values are set to $\gamma_T{=}0.1$ and $\gamma_\ell{=}0.1$. From the current state, it is possible to change the node (switching to more capable n_1), model (switching quicker-converging m_1), both, or neither; such actions are represented (resp.) by solid green, solid purple, dashed blue, and dotted black edges in the figure. Different combinations of possible switches yield different combinations of loss and elapsed time, only one of which – the bottom, pink vertex – is feasible, hence, connected to Ω.

Step 2: Feasible paths. Next, PACT uses the expanded graph to identify a set of paths deemed *feasible*; the first edge of such paths represents a feasible action. To mitigate the impact of potential errors in the loss estimation (which in principle may jeopardize feasibility), the expanded graph is built using the *robust* estimators of the loss variation, which guarantees that all paths landing at a feasible node are, indeed, feasible with high probability. Thus, using function FindFeasiblePaths in

Algorithm 3 Finding feasible paths

```
1: function FindFeasiblePaths
2:     v_curr ← (k, m, n, ℓ, T)
3:     P ← ∅                                              ▷ feasible paths
4:     for all v: (v, Ω) ∈ E do
5:         p ← shortestPath(v_curr, v)
6:         P ← P ∪ {p, weight= Σ_{e∈p} weight[e]}
7:     return P
```

Alg. 3, PACT seeks for paths that (i) start from the current state, and (ii) arrive to a feasible state, i.e., to a vertex connected to Ω. Specifically, for each vertex v corresponding to a feasible state, it determines the shortest path (Line 5) from the current state v_{curr} to v. Such paths are collected in set $\mathcal{P}$ and associated with a weight corresponding to the sum of weights (i.e., energy consumption) of their edges.

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

For every path $p \in \mathcal{P}$, scores are computed in the ChooseAction function in Alg. 4. The opportunity factor, $\texttt{opp} \geq 1$, is given by the ratio of (i) the sum of the expected loss to (ii) the sum of the robust loss associated with the edges in p (Line 9). The intuition is to make it easier to choose actions with a good expected loss, since

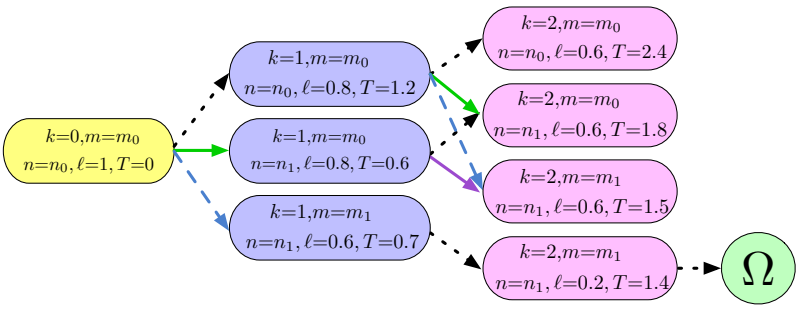


Fig. 3.5 Example of the PACT expanded graph, with resolution values γ_T=0.1 and γ_ℓ=0.1, learning target $\ell^{\max}$=0.25, and time limit $T^{\max}$=1.5. Edge colors denote switches across subsequent epochs: node only (solid green), model only (solid purple), both (dashed blue), neither (dotted black).

Algorithm 4 Choosing the next action

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

the robust estimator may be too pessimistic. As for the risk factor, its high-level purpose is to avoid undoing decisions. To this end, PACT seeks for paths on the expanded graph that lead from the first node of p, to a vertex $\overline{v}{\in}\overline{\mathcal{V}}$ associated with the current model m (Line 10), and thence to Ω. The risk factor, $\texttt{risk}{\geq}1$, associated with path p is then computed in Line 12 as the ratio of the minimum among the weights of such paths to the weight of p (defined in Alg. 3).

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

3.5.1 Problem and algorithm analysis

Property 5. *The problem of optimizing (3.5) is NP-hard.*

The proof is based on a reduction in polynomial time from the generalized assignment problem (GAP) [67], which is known to be NP-hard. Furthermore, we prove that:

Property 6. *PACT's time complexity is polynomial.*

Proof. PACT's complexity is given by the sum of the complexity of Alg. 2–Alg. 4. In Alg. 2, the first loop is run at most $|\mathcal{V}|{=}MN\left\lceil\frac{T^{\max}}{\gamma_T}\right\rceil^2\left\lceil\frac{\ell(0)}{\gamma_\ell}\right\rceil$ times, where $M{=}\,|\mathcal{M}|$ and $N{=}\,|\mathcal{N}|$, i.e., denotes the number of the predefined sets of nodes in $\mathcal{N}$. The second loop in Alg. 2 is run at most $|\mathcal{V}|MN$ times. Alg. 3 computes at most $|\mathcal{V}|^2$ shortest paths, each of which (e.g., using Dijkstra's algorithm [102]) incurs a polynomial complexity. Alg. 4 iterates over set $\mathcal{P}$ of feasible paths, whose number cannot exceed $|\mathcal{V}|$ (as per Alg. 3, Line 4). Thus, all contributions to PACT have polynomial complexity, which proves the thesis. □

Importantly, Property 6 concerns the *worst-case* time complexity of PACT, which in practice has substantially lower complexity. In particular, the shortest-path routines used in Alg. 3 and Alg. 4 have been heavily optimized, and perform very efficiently in practice [102].

We can further prove that the space complexity of PACT's most complicated part, i.e., the CreateExpandedGraph procedure in Alg. 2, does not exceed that of its output, i.e., the expanded graph itself.

Property 7. *The space complexity of Alg. 2 is* $|\mathcal{V}|^2MN$*, with* $|\mathcal{V}|{=}MN\left\lceil\frac{T^{\max}}{\gamma_T}\right\rceil^2\left\lceil\frac{\ell(0)}{\gamma_\ell}\right\rceil$

Proof. The proof follows by inspection of Alg. 2. First, we remark that no data structures are created within the algorithm. Then we observe that at most one vertex is created every time the first loop in the algorithm is run, and at most one edge is created every time the second loop is run. Considering (see also the proof in Property 6) that the first loop runs at most $|\mathcal{V}|{=}MN\left\lceil\frac{T^{\max}}{\gamma_T}\right\rceil^2\left\lceil\frac{\ell(0)}{\gamma_\ell}\right\rceil$ times, and the outer loop at most $|\mathcal{V}|MN$ times, then the thesis holds. □

The intuitive meaning of Property 7 is that the space complexity of Alg. 2 does not exceed that of its output, further supporting the suitability of PACT even to large-scale, complex scenarios.

At last, we prove the following property about how good PACT's solutions are at minimizing the objective in (3.5).

Property 8. *If predictions are exact, their time horizon is sufficiently long, and all* $\Delta\ell$ *and* ΔT *values are integer multipliers of* γ_ℓ *and* γ_T*, then PACT is optimal.*

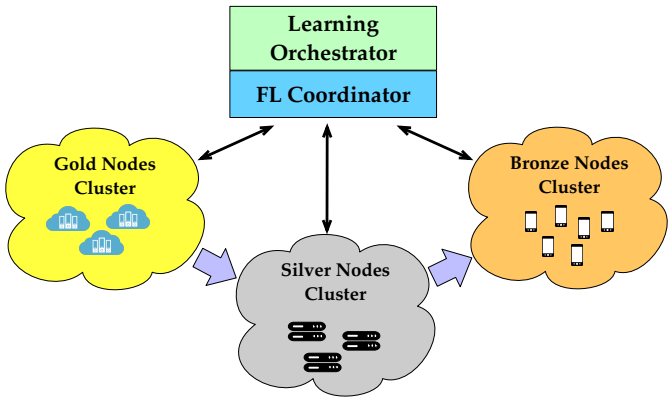


Fig. 3.6 Federated scenario. Nodes of the same category form a cluster, within which learning is performed in parallel, employing FedAvg; the aggregation of the models is then performed by an FL coordinator. Over the clusters, instead, the learning procedure takes place sequentially.

Proof. The proof comes from inspection of Alg. 2–Alg. 4, which consider all possible options (e.g., all vertices in Alg. 3, all paths in Alg. 4), hence, no feasible solutions are ignored. Further, the shortest-path problem in Alg. 3 and Alg. 4 can be efficiently solved to the optimum. If the hypothesis holds, then the ceiling operators in Alg. 2 (Line 17 and Line 18) have no effect, hence, there is no possible source of suboptimality. □

An important consequence of Property 8 is that, by varying γ_ℓ and γ_T, we can effectively trade off how close to the optimum the solution gets with PACT's time complexity.

3.6 Numerical results

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

- a *sequential learning* scenario, like the one depicted in Fig. 3.1, where models are passed among individual nodes;
- a *federated learning* scenario, as depicted in Fig. 3.6, where *clusters* of nodes are employed instead.

Table 3.1 Example actions and their effects in the sequential learning and federated learning scenarios

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

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

Importantly, the PACT methodology – e.g., building the expanded graph and finding a shortest path therein – works unmodified in both scenarios. Needless to say, the available actions (i.e., node-selection and model-switching decisions), as well as their effects on the learning quality and energy consumption, are different and are estimated as discussed in Sec. 3.4. An example of possible decisions and their effect is presented in Tab. 3.1.

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

Learning always starts with the gold node training the full model. Then either one or two pruning steps (i.e., two or three models) are considered, with pruning being performed as described in Sec. 3.2. In the first case, after K_1 epochs, the model is pruned with pruning ratio F_1 and handed over to the silver or the bronze

node, which continues the training. If instead two pruning steps are performed, then the training at the silver node is interrupted after K_2 epochs, after which the model is pruned with fraction F_2 and sent to the bronze node. Two convolutional DNNs are considered, namely, VGG-19 and AlexNet [2].

Importantly, different combinations of F_1 and F_2 correspond to different elements of the models set $\mathcal{M}$, hence, setting those values is equivalent to selecting the models to use. Recall that model pruning also implies switching to a different node, so that more complex models are always matched with more capable nodes. Specifically, the training time and energy values used for the gold, silver, and bronze nodes reflect (resp.) the capabilities of NVIDIA Ampere A100 [71], NVIDIA RTX A4000 [72], and Raspberry Pi's Videocore 6 [103] GPUs. Finally, for simplicity, we set a very long time limit of $T^{\max}$=1,000 time units. Notice how the energy cost associated with nodes is incurred only *while* the nodes themselves are used, i.e., the energy consumption of idle nodes is neglected [93, 94].

Federated scenario. In the FL scenario, we replace individual nodes with *clusters*, each including two nodes and a learning coordinator. The latter performs model averaging after each epoch following the FedAvg algorithm [7], on the grounds that it is the vanilla approach to FL, hence, provides the easiest-to-replicate results. Note that the FL coordinator (running FedAvg) and the learning orchestrator (running PACT) are two different logical roles, which may not necessarily be taken on by the same physical node.

3.6.1 Loss prediction implementation

As described in Sec. 3.4, we use a DNN and an LSTM to estimate (resp.) λ^{run} and λ^{change}. To capture the difference between different settings, we train separate models for each of our scenarios, namely federated and sequential; further, in the latter scenario, we train separate models for the cases when the VGG and AlexNet networks are used. Importantly, each model is trained considering all nodes – or clusters of nodes in the federated case – and different combinations of K_1, K_2, F_1 and F_2, namely $K_1, K_2 \in [1, 5, 15, 25, 40, 60]$ and $F_1, F_2 \in [0.25, 0.5]$; this allows us to build estimators that work well over different models and nodes and, hence, their associated datasets, which are fed as input to the estimators, as explained in Sec. 3.4.

Table 3.2 Loss prediction: mean and standard deviation for all scenarios

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

Because we are interested in estimating the whole distribution of the loss, we employ a customized loss function given by the summation of the mean square error (MES) and a *tilted loss term* [104] ensuring that all quantiles are correctly estimated.

The prediction quality achieved is reported in Tab. 3.2, summarizing the prediction metrics we consider, namely: (i) the mean absolute error (MAE); (ii) the mean interval length (MIL), i.e., the average width of the prediction interval; (iii) the interval coverage percentage (ICP), i.e., the fraction of true values falling within the relative prediction interval. The latter two metrics are linked with the quality of quantile predictions. From the table, it clearly emerges that the loss prediction is very good in all scenarios, a further evidence of the validity of the PACT solution strategy and the underlying intuition.

3.6.2 PACT performance

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

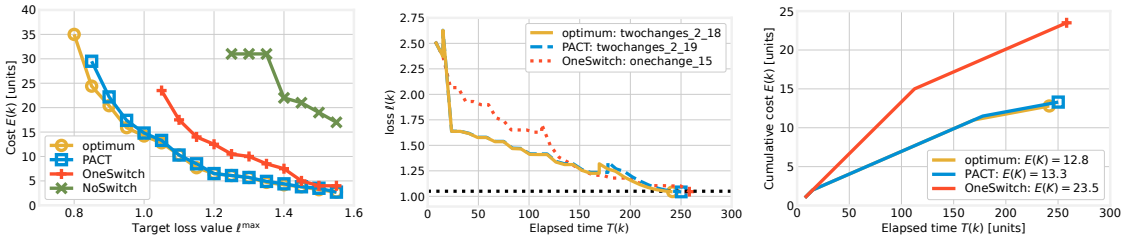


Fig. 3.7 PACT and benchmark strategies for sequential scenario with VGG: cost for different values of $\ell^{\max}$ (left); evolution of loss (center) and cost (right) when $\ell^{\max}$=1.05.

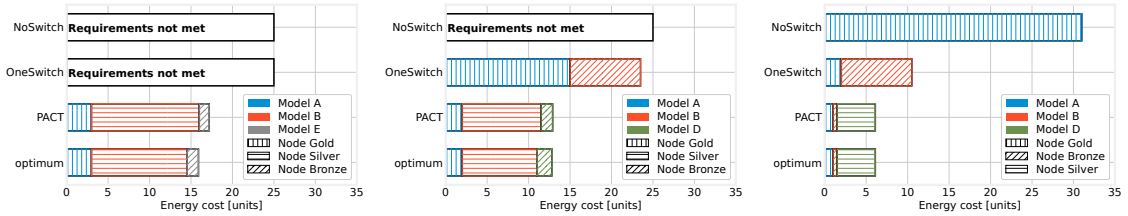


Fig. 3.8 PACT vs. benchmark strategies for sequential scenario with VGG: energy cost incurred by using different models when $\ell^{\max}$=0.95 (left), $\ell^{\max}$=1.05 (center), $\ell^{\max}$=1.25 (right).

works [87, 105, 106] envision pruning once, hence, their performance would be represented by *OneSwitch*.

Sequential scenario, VGG model. First, we evaluate PACT's effectiveness, i.e., how the cost (i.e., the consumed energy $E(K)$) it yields compares to that of the benchmarks. To this end, Fig. 3.7(left) shows the cost associated with each strategy, for different loss targets $\ell^{\max}$. We can observe that the NoSwitch strategy is the worst one and does not allow reaching a low value of $\ell^{\max}$, as only the full model is used. Recall that the full model is trained by the gold node, which has many data samples, but distributed in an extremely unbalanced manner, as 4 out of 10 classes are highly under-represented. Excluding the NoSwitch strategy, when $\ell^{\max}$ is relatively high, all strategies result in very similar performance; on the other hand, they diverge as $\ell^{\max}$ decreases, i.e., as the conditions become more challenging. In particular, PACT closely matches the optimum, to the point that the corresponding curves almost overlap, and outperforms OneSwitch solution – which is the approach followed in most state-of-the-art works. Switching across *multiple* models and nodes is indeed beneficial when learning constraints are tight.

Fig. 3.7(center) depicts the time evolution of $\ell(k)$ for $\ell^{\max}$=1.05. Note that the peak due to the loss variation λ^{change} incurred upon model switching is not always

present, as the testing loss can decrease even when switching the model. This is especially true when the first switching occurs early, as depicted by the two curves with K_1=2, relative to the PACT and the optimal solutions. Remarkably, PACT makes virtually the same decisions as the optimal policy, i.e., performs the model switching at (almost) the same times. OneSwitch can only switch once, hence, does so later. Interestingly, all the strategies achieve the learning target at almost the same time. However, we recall that *cost* is the optimization objective (3.5), while time is a mere constraint. Accordingly, Fig. 3.7(right) highlights how the optimum indeed takes slightly shorter than PACT to reach the objective and does so at a (marginally) lower cost (see the position of the last marker on the y-axis). On the other hand, OneSwitch solution, despite taking a comparable time, requires much more energy to reach $\ell^{\max}$.

Fig. 3.8 sheds further light on how different strategies use the network infrastructure. Plots therein show how much energy is spent running each of the distinct models under the different strategies; different plots correspond to different values of $\ell^{\max}$, that are 0.95, 1.05, and 1.25 (resp.). Consistently with Fig. 3.7(right), when $\ell^{\max}$ is high, NoSwitch requires the larger amount of energy, i.e., higher cost, while PACT and optimum incur almost the same cost. As for OneSwitch, the energy cost is between the one of PACT and NoSwitch. As shown in Fig. 3.8(center), for a medium value of $\ell^{\max}$, NoSwitch cannot achieve the target loss value. Thus, the OneSwitch solution requires the highest amount of energy, while PACT and optimum decisions entail a very similar cost, even if the decisions are a bit different, as depicted in Fig. 3.8(center) by the different costs of models B and D. When $\ell^{\max}$ is low, as in Fig. 3.8(left), the difference between PACT and OneSwitch emerges more clearly: PACT requires more energy and takes different decisions, employing model B more. Further, in this case, also OneSwitch solution cannot achieve the desired $\ell^{\max}$.

Next, we assess the impact of γ_ℓ and γ_T, which control the trade-off between PACT's complexity and representation's granularity. Fig. 3.9(left) shows that a larger value of γ_ℓ does decrease PACT's performance: indeed, the minimum value of $\ell^{\max}$ that can be achieved significantly increases with γ_ℓ. However, even increasing γ_ℓ by an order of magnitude, PACT still outperforms OneSwitch for large $\ell^{\max}$, while with lower $\ell^{\max}$ it is comparable to, or slightly worse than, OneSwitch.

Fig. 3.9(center), referring to the case $\ell^{\max}$=1.05, provides some insight on *how* a higher γ_ℓ affects the decisions made by PACT. Specifically, the higher the value

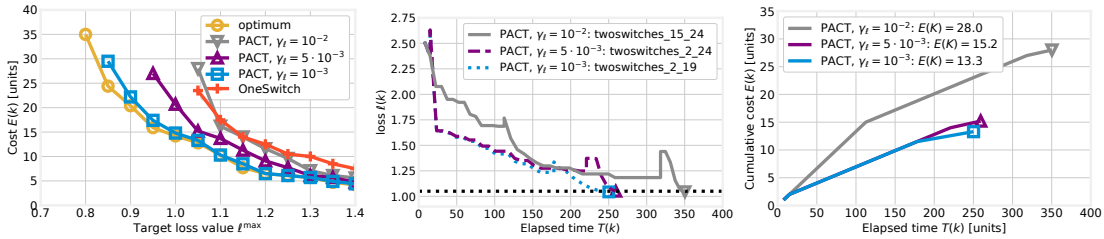


Fig. 3.9 Impact of γ_ℓ on PACT's performance for sequential scenario with VGG: energy cost for different values of $\ell^{\max}$ (left); evolution of loss (center) and cost (right) when $\ell^{\max}$=1.05.

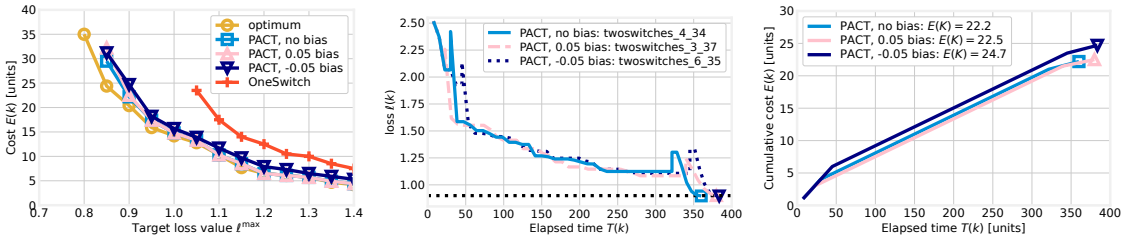


Fig. 3.10 Impact of quality of $\hat{\lambda}^{\text{run}}$ for the sequential scenario with VGG: PACT's energy cost for different values of $\ell^{\max}$ (left); loss (center) and cost (right) evolution for $\ell^{\max}$=0.9.

of γ_ℓ, the later switches are made. The reason lies in Line 17 of Alg. 2, and more exactly in the ceiling operator therein. Increasing γ_ℓ leads to overestimating the loss resulting from a particular action, hence, to assume that further gains could be made under the current model, while that is not the case. For the same value of $\ell^{\max}$, Fig. 3.9(right) highlights how these later switches result in a higher cost and time.

Finally, we further assess how well PACT can deal with loss estimation errors, by adding a *bias* to the prediction output for the first model, namely, the full one. Fig. 3.10(left) shows that positive and negative biases yield similar performance decrease. Also, PACT outperforms OneSwitch even in the presence of a bias. Fig. 3.10(center), referring to the case $\ell^{\max}$=0.9, shows how biases on the loss variations prediction influence PACT's decisions. Consistently with Fig. 3.9(center), underestimating the full model's performance leads to a later switch, while overestimating it has the opposite effect. It is also worth noting the times of the *second* switch: PACT can potentially compensate for the misguided decisions it made earlier. This happens, for instance, when the bias is equal to 0.05, as K_1 and K_2 are respectively lower and larger w.r.t. the case with no bias. Fig. 3.10(right) underlines, similarly to Fig. 3.9(right), that adding a bias term leads to higher cost and time.

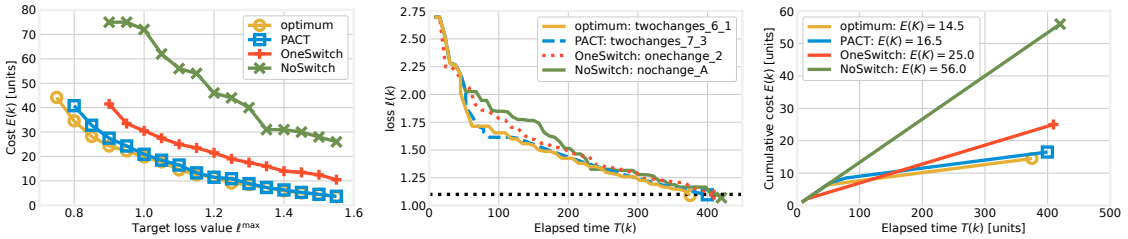


Fig. 3.11 PACT and benchmark strategies for sequential scenario with AlexNet: cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max}$=1.1.

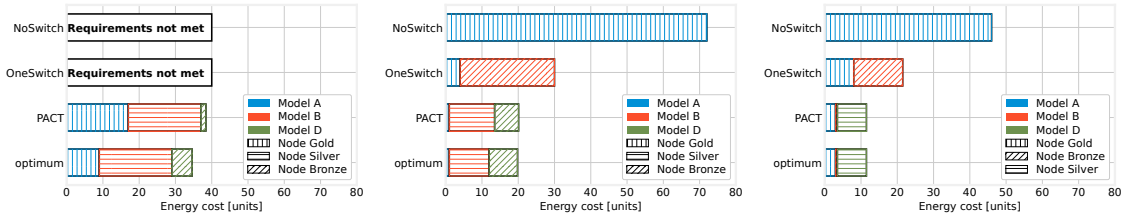


Fig. 3.12 PACT vs. benchmark strategies for sequential scenario with AlexNet: energy cost incurred by using different models when $\ell^{\max}$=0.8 (left), $\ell^{\max}$=1 (center), $\ell^{\max}$=1.2 (right).

This is a very important result, as it highlights how PACT is robust to possible errors in estimation techniques such as that demonstrated in Sec. 3.4.

Sequential scenario, AlexNet model. Fig. 3.11(left) shows the cost versus the target loss, for PACT and for each of the benchmark solutions. As in Fig. 3.7(left), PACT decisions are close to the optimal ones and outperform the other solutions.

Fig. 3.11(center) shows the loss trend as a function of the elapsed time when setting ℓ^{max}=1.1: the PACT solution takes slightly longer than the optimal one, but still less than the other two solutions. Nevertheless, in general, the time necessary to achieve the desired ℓ^{max} is similar for all the solutions. On the other hand, as depicted by Fig. 3.11(right), the incurred cost changes among the different strategies: PACT's cost is slightly higher than the optimal one, but lower than the energy cost entailed by OneSwitch and NoSwitch.

Fig. 3.12, similarly to Fig. 3.8, shows how much the different nodes are exploited, considering three different values of $\ell^{\max}$, respectively 0.8, 1, and 1.2. When the requirements are stricter, i.e., $\ell^{\max}$ is lower, the NoSwitch and OneSwitch solutions cannot find any feasible solution and generally the energy consumption increases.

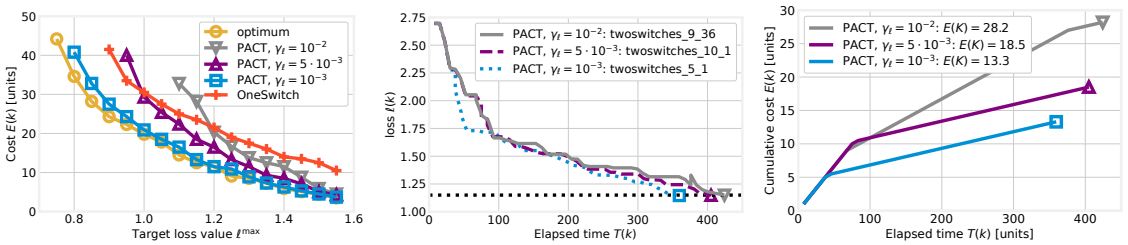


Fig. 3.13 Impact of γ_ℓ on PACT's performance for sequential scenario with AlexNet: energy cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max}$=1.15.

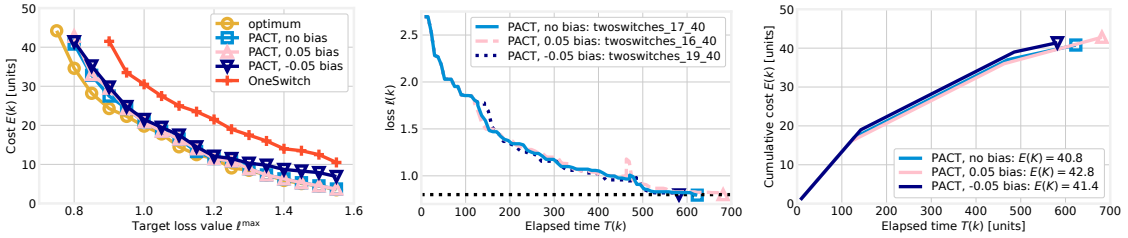


Fig. 3.14 Impact of quality of $\hat{\lambda}^{\text{run}}$ for sequential scenario with AlexNet: PACT's energy cost for different values of $\ell^{\max}$ (left); loss (center) and cost (right) evolution for $\ell^{\max}$=0.8.

Again, when the constraints are tighter, PACT requires more energy than the optimal solution.

Fig. 3.13(left) shows the impact of γ_ℓ: when $\ell^{\max}$ decreases, the effect of increasing γ_ℓ is noticeable and it may be not possible to reach the same $\ell^{\max}$ value as the optimum. In this case, unlike in the VGG results of Fig. 3.9(left), when $\ell^{\max}$ decreases, OneSwitch outperforms first the solution obtained with γ_ℓ=10^{-3} and then the one with γ_ℓ=$5 \cdot 10^{-3}$. Regarding Fig. 3.13(center) and Fig. 3.13(right), we can draw the same conclusions as for Fig. 3.9(center) and Fig. 3.9(right).

Fig. 3.14(left) underlines the impact of γ_T: also in this case, PACT outperforms the OneSwitch solution in the presence of a bias. Unlike Fig. 3.10(center), even if the effect of a bias on K_1 decision is the same, a negative bias value leads to a solution requiring less time; however, as depicted by Fig. 3.14(right), the energy cost always increases when applying a bias.

Federated scenario, VGG model. Fig. 3.15, Fig. 3.16, Fig. 3.17, and Fig. 3.18 depict the results obtained in the federated setting. Generally, it is possible to draw the same conclusions as for the sequential scenario. Also, Fig. 3.18(center) shows that for $\ell^{\max}$=1.1 applying a bias equal to 0.05 does not influence PACT's decision.

Still considering the same bias value, looking at Fig. 3.18(left), we can notice that the trend is not monotonic. Indeed, when $\ell^{\max}$=1.15, the incurred cost is higher than for $\ell^{\max}$=1.1. This result might be counterintuitive, but the explanation is straightforward. When $\ell^{\max}$=1.15, applying a bias equal to 0.05 leads to a very inefficient K_1 choice, as it is necessary to spend a very high cost to achieve the desired loss value. Applying the same bias when $\ell^{\max}$=1.1, instead, it has a much lower effect on the decision and, thus, on the cost, which is lower than the one in the previous case.

3.7 Related Work

Our work is related to four main research areas, namely: approaches to transform an ML model into a different one, hybrid learning strategies that combine different techniques, resource-aware distributed ML, and distributed learning characterization.

Model switching and compression. It is often necessary to transform a pre- or partially-trained model into a simpler and/or smaller one, preserving (as much as possible) the learning accuracy. The two most popular techniques for model compression are knowledge distillation (KD) and pruning. Both techniques are used to achieve *model compression*, i.e., to obtain small-size models exhibiting the same learning performance as a larger one. KD [87] is a family of learning techniques predicated on the transition from a larger *teacher* model to a smaller (hence, simpler) *student* one. The student model does not learn directly from data, but mimics the behavior of the teacher model. Such learning exploits not only the decisions of the teacher model, but also the so-called *dark knowledge* embedded in its parameters. As shown in [107], additional properties such as regularization are also transferred from teacher to student models. Several works combine KD with deep reinforcement learning (RL), to learn the *value* of each *action*, as the actions themselves are taken. Studies focus on task generalization [105], in which models are transferred across nodes in charge of different tasks, and knowledge is distilled from a task to another. In the same context, [106] focuses on the heterogeneity of data at different nodes, and on mitigating its impact on learning performance. [85] uses KD to achieve *domain adaptation* in scenarios where no data from the source domain, but only a model trained therein, is available. A different aspect of KD is tackled by [108, 109], aiming at distilling multiple single-task policies into a single, multi-task policy.

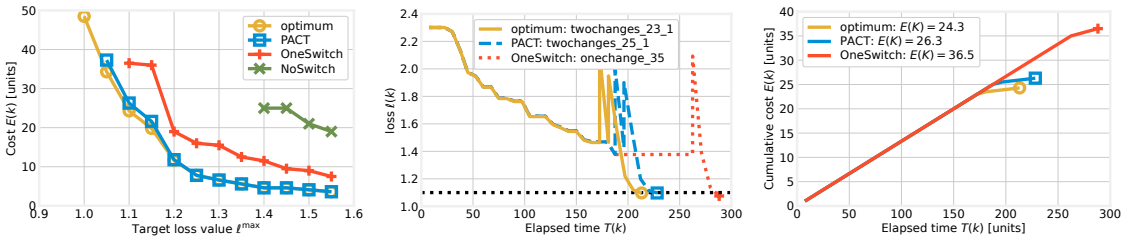


Fig. 3.15 PACT and benchmark strategies for the federated scenario with VGG: cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max}$=1.1.

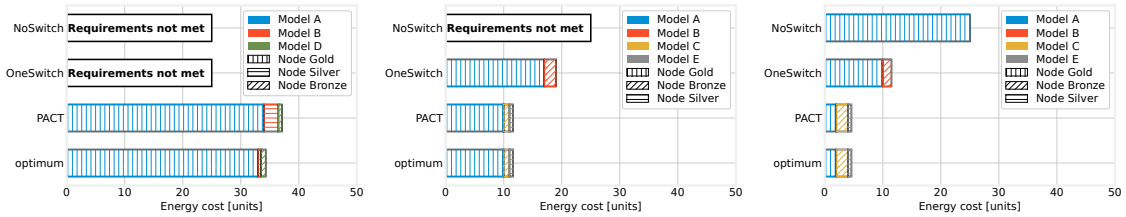


Fig. 3.16 PACT vs. benchmark strategies under the federated scenario with VGG: energy cost incurred by using different models when $\ell^{\max}$=1.05 (left), $\ell^{\max}$=1.2 (center), and $\ell^{\max}$=1.4 (right).

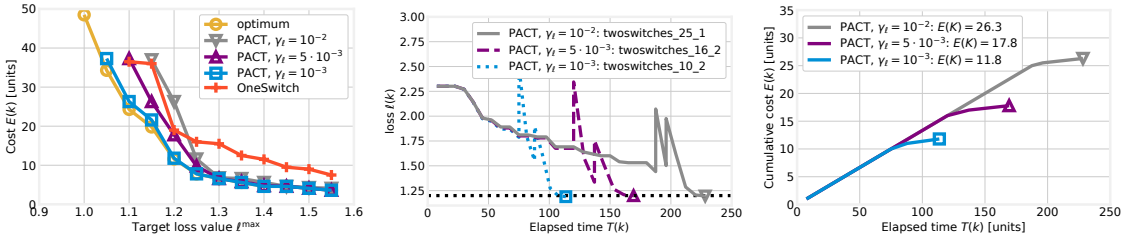


Fig. 3.17 Impact of γ_ℓ on PACT's performance for the federated scenario with VGG: energy cost for different values of $\ell^{\max}$ (left); evolution of the loss (center) and cost (right) when $\ell^{\max}$=1.2.

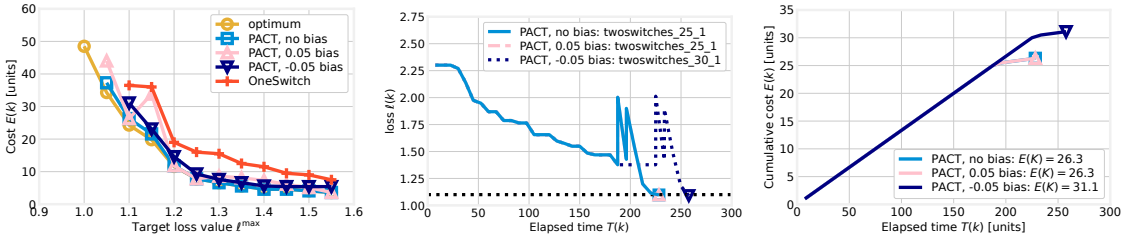


Fig. 3.18 Impact of quality of $\hat{\lambda}^{\text{run}}$ for the federated scenario with VGG: PACT's energy cost for different values of $\ell^{\max}$ (left); loss (center) and cost (right) evolution for $\ell^{\max}$=1.1.

Model pruning follows instead the more direct approach of removing some of the coefficients from a model, thus reducing its size and complexity. It is based upon the so-called *lottery ticket* hypothesis [110], assuming that very few of the parameters of a model actually impact its performance. Once such parameters can be identified, the others can be *pruned* away, resulting in a DNN with the same performance as the original one, but a much lower complexity. Several works have addressed iterative pruning where every few epochs the weights expected to least affect performance are pruned [111, 112]. A hardware-friendly scheme is proposed, e.g., in [113], with the aim to execute DNNs on low-end hardware in near real time. [92] brings privacy into the picture, highlighting how simplistic pruning may expose user data. A very effective technique is also *structured* pruning [86], which removes whole parts of a DNN (e.g., rows or columns of the parameter matrix) instead of individual ones, thus offering better efficiency, than plain pruning, at the cost of more complex decisions to make.

Pruning can be combined with KD, as envisioned in [114], where it is shown that pruned networks are easier to distill. A related problem is that pruning affects the performance of different layers to a different extent. To cope with this, [115] proposes using pruning on some DNN layers of the ResNet architecture, and compressing the others via KD. Similarly, in [116] each layer is pruned and distilled independently and in a parallel, to achieve faster distillation. Finally, recent work [117] proposes the use of RL to control pruning. In a similar spirit, [118] automatically looks for the best pruning ratio to use at every layer of a DNN, so as to simplify (hence, speed-up) learning without hurting accuracy.

Hybrid approaches. Combining different models towards a single learning task is a fairly uncommon approach. Some works explore how to alternate distributed learning schemes such as Split Learning (SL), FL, and KD. An example is [119], which splits the DNN architecture into head and tail, and replaces the former with its distilled version. [120] seeks to reduce the network delay incurred by FL by performing communication and local learning concurrently, at the price of the global model being behind local ones by several epochs. In a similar setting, [121] optimizes the computation, communication, and cooperation aspects of FL in resource-constrained scenarios. [122] leverages RL to identify the best split of a learning task (e.g., the layers of a DNN) across the available network nodes.

A related issue is *early exit*, aiming at making inference faster by skipping some of the DNN layers if a certain outcome, e.g., a certain classification decision, can be reached with high certainty. This is achieved [123] by giving the DNN a Y-shaped topology, with one branch being a lot simpler (e.g., with a smaller number of neurons) than the other. The simpler branch is ran first and, if the results are decisive enough for a certain sample, then the more complex branch can be altogether dispensed with. [124] targets highly heterogeneous scenarios where learning *one* model for all devices may indeed be suboptimal, and proposing a *personalized learning* where a different model is trained at each device; layers that are common among models are then trained in an FL fashion.

Resource-aware distributed ML. A major concern of distributed ML is *the nodes heterogeneity*, in terms of data quality, data quantity, available resources, and connectivity. In the context of FL, several works focus on *selecting* the participating nodes, accounting for their speed [29, 125], quantity [124, 125, 124, 126] and quality [126, 127] of local data, the speed and reliability of their network [29, 120] as well as trust [128]. The basic trade-off balances the need to learn more during each epoch (achieved primarily by adding more data from more nodes) with the need to shorten the epoch duration. Some studies seek to assign a trust score to each node, combining its speed and the likelihood to report inaccurate information [128]. Recent studies [129] have established a benchmark system, including datasets and helper Python classes, to easily compare FL strategies. Emulated *testbeds* for distributed ML are also emerging. A prominent example is [130], allowing the study of novel ML schemes and strategies performance over real, network-edge-class hardware. A related trend is represented by novel datasets, with tried-and-tested options like CIFAR [62] complemented by more challenging alternatives; e.g., [131] describes a dataset containing hundreds of thousands of images of numbers taken from Google Maps.

Other works, e.g., [94, 132], target a more general scenario, where DNN layers can be run, and possibly be duplicated, at different nodes. This requires balancing the opportunity to use fast learning nodes with the network delays resulting from moving data between nodes. Both works jointly consider learning time and energy consumption: in [94] they are part of a combined objective, while in [132] the former is a constraint and the latter the sole objective. Interestingly, recent work (e.g., [133]) has aimed at creating energy-efficient DNN architectures, offering better trade-offs between energy efficiency and learning effectiveness. For instance, [133] presents

variants of the MobileNet and ResNet architectures, achieving good performance with a fraction of the parameters (hence, training time) of their counterparts. Similarly, [134] proposes a family of scalable DNN architectures for sound event detection, suitable for scaling depending upon the available resources. In the same spirit, the authors of [78] seek to improve FL by dividing models into *client-* and *server*-side parts, with the former providing personalized results for local datasets, and the latter achieving high accuracy for out-of-distribution data. A different approach is adopted in [135], dropping the usage of *deep* NNs and finding instead that single-layer, *wide* NNs perform as well as their counterparts. This has an impact on how easy it is to run the learning task in parallel, as wide NNs have fewer dependencies between parameters and are thus easier to split across nodes.

Distributed learning characterization. Early work [28] studies the convergence of distributed learning, identifying the latent trade-off between involving more nodes (which means that fewer epochs are needed to converge) and exploiting fewer, faster nodes (which makes individual epochs shorter). The experiments in [100] report a power-law behavior, with the exponent depending upon the quantity of data, and the model architecture shifting the error, but not reducing the exponent itself. Other works focus on FL and derive exponential bounds on the loss [136, 35]. Some studies focus on the overparameterized regime of DNNs, investigating, e.g., how many parameters are needed for a network to be overparameterized [137, 97], Specifically, [97] finds that the number of required parameters grows quadratically with the data size (number of samples and complexity thereof). Studies focusing on KD are more rare. Examples include [138], which models the teacher-to-student translation as a price to pay on the loss, and [139] that provides a per-iteration characterization of KD.

Finally, we mention that a preliminary version of this work has been presented in our conference paper [140], where however only sequential learning was addressed and the results were obtained using a single DNN.

3.8 Conclusions

In this chapter, we addressed the problem of optimizing the strategy for the distributed training of DNN models, by making joint decisions about (i) model switching (e.g., pruning), (ii) data selection, and (iii) node and resource allocation. Our objective is

to minimize the energy cost, subject to learning quality and time constraints. In view of the problem NP-hardness, we proposed an algorithmic framework called PACT, following an ADP approach and leveraging both theoretical results and the outcome of previous decisions to predict the future training evolution. PACT has polynomial worst-case complexity, and the decisions it makes can be arbitrarily close to the optimum. We have further shown, through extensive numerical evaluation, that PACT outperforms state-of-the-art approaches and closely matches the optimum.

In the next chapter, we will address the issue of the lack of incentives for learning nodes in the context of Decentralized Learning, a relevant challenge in cooperative settings where nodes cannot be forced to use their computational resources to train or fine-tune an ML model. To promote collaborative behavior among nodes, we propose a game-theoretic method that accounts for the heterogeneity of nodes' resources and models to train.

Chapter 4

Generosity Pays Off: A Game-Theoretic Study of Cooperation in Decentralized Learning

Decentralized learning, a paradigm enabling the training of Machine Learning (ML) models using multiple nodes, is gaining momentum, as it (i) improves data privacy and (ii) permits to leverage the computational capabilities of a wide set of nodes, thus being an excellent fit for the support of edge intelligence applications. However, such nodes, like users' smartphones or vehicles, cannot be *forced* to participate in the learning process, and incentivizing them to do so is one of the foremost challenges of decentralized learning. To address this issue, we propose GENIAL – a game-theoretic approach, based upon generous games, to promote cooperation among user nodes for training or fine-tuning ML models. By allowing such nodes to be (moderately) *generous*, i.e., to contribute to decentralized training processes more often than what would be convenient for them in the short term, GENIAL leads to a Nash equilibrium where all nodes cooperate. Importantly, such equilibrium is also proven to converge to the Pareto optimal operating point that ensures a fair treatment to all nodes. Our theoretical findings are supported by numerical experiments, which further underline the effectiveness, and the benefits for rational nodes, of being generous in decentralized training.

Part of the work described in this chapter has already been published in **Di Giacomo, G.**, Malandrino, F., Chiasserini, C. F. (2024, June). Generosity Pays Off: A Game-Theoretic Study of Cooperation in Decentralized Learning. In 2024 IEEE International Conference on Communications Workshops (ICC Workshops) (pp. 105-110). IEEE..

4.1 Introduction

The decentralized learning paradigm [141, 142] has emerged as a powerful alternative to the traditional centralized learning approach. Contrary to the latter, the decentralized learning paradigm leverages the computational power of the participating nodes, which locally train the model by using their own data, thus ensuring data locality and privacy [142], and then exchange such model till convergence is reached.

Such a learning paradigm is especially relevant in the case where nodes at the edge and/or far-edge of the network have to be involved for the training or the fine-tuning of ML models, owing to the diversity of the data they own and the need to keep such data local. However, in many real-world scenarios, edge and far-edge nodes belong to different entities or are user devices, hence, they are under no obligation to participate in the learning and will do so only if advantageous to them. In game-theoretic terms, they are *rational*, and a proper incentive mechanism must be in place to make them participate [143].

To address this issue, we propose an algorithm called GENerous Incentive Algorithm for Learning (GENIAL), which is based on the classic Generous Tit-for-Tat strategy [144, 145] (GTFT). Under GTFT strategies, nodes start by being willing to cooperate (hence, "generous"), and then keep doing so only if they observe that other nodes cooperate as well (hence, applying "tit-for-tat"). We show that GENIAL yields a Nash equilibrium, i.e., a profile strategy where no player gains some profit by unilaterally modifying its strategy. Furthermore, we prove that such an equilibrium is Pareto optimal and fair, i.e., it is not possible to make a player better off without negatively affecting another one.

Intuitively, reaching a Pareto-optimal Nash equilibrium means that the system operates in an efficient manner, and such a result is reached *in spite* of the fact that all nodes solely pursue their own interest. GENIAL is able to account for the most

relevant aspects of ML training, most importantly, that nodes may belong to different *classes*; the class of a node dictates the complexity of the models each node needs to train, its target learning quality, and its available computational resources. It follows that decisions about whether or not to cooperate may have different consequences, i.e., impact on the nodes' resource consumption, each time they are made.

In the rest of the chapter, Sec. 4.2 introduces the system model, while Sec. 4.3 formulates the problem of cooperative decentralized learning as a maximization problem of the payoff received by the nodes. Sec. 4.3 derives the Pareto-optimal operating points. Sec. 4.4 proposes the GENIAL algorithm as incentive mechanism, and it shows that GENIAL attains the Pareto-optimal Nash Equilibrium. The performance of GENIAL is shown in Sec. 4.5. Finally, Sec. 4.6 discusses some related work, while Sec. 4.7 draws our conclusions.

4.2 System Model

We consider a learning orchestrator at the edge of the network, which assists a cluster $\mathcal{N}$ of N nodes with the training or the fine-tuning of ML models. For simplicity, the cluster topology is assumed to be fully connected. Further, each node belongs to one of K classes, with the number of nodes in class k being denoted with N_k. All nodes in class k are associated with an average and a maximum computing capability constraint, denoted respectively by $\bar{\chi}_k$ and $\chi_k^{\max}$; without loss of generality, we assume: $\bar{\chi}_1 < \bar{\chi}_2 < \ldots < \bar{\chi}_K$ and $\chi_1^{\max} < \chi_2^{\max} < \ldots < \chi_K^{\max}$. Also, each class k is associated with a set of models $\mathcal{M}_k$ that the nodes in the class may need to train or update: the complexity of the models in $\mathcal{M}_k$ is denoted with m_k. The quantities $\bar{\chi}_k$ and $\bar{\chi}_k^{\max}$ are measured in millions of floating point operations per second (MFLOPS), while the complexity m_k indicates the number of floating point operations (FLOPs) to process one sample during training.

A node asking for help is referred to as *requester*, while a node accepting to help train a model is called *learner*. Whenever a training process is concluded, the orchestrator selects the next node to start a new training process among the candidate requesters with probability $1/N$, so that only one training session at the time can be performed.

We denote by $h{+}1$ the minimum number of nodes needed to train the model at hand, where one of such nodes is the requester itself. Fixing *a priori* the value of h, we can compute the minimum number of training samples d_k that each node should use to train a model of complexity m_k, such that the target learning quality is achieved. More specifically, in, e.g., [100], it is empirically found that the generalization error (i.e., the test loss value) exhibits a power-law improvement with respect to the training set size. The power-law parameters, however, depend on the specific DNN and dataset at hand; as also envisioned in [146], to estimate such parameters, one could run a small-scale profiling for a specific range of test losses. Thus, given the target test loss value (which is lower than the values used for the profiling), it is possible to derive the minimum number of required training samples D_k for a model of complexity m_k. Given D_k and the value of h, which are inputs to our problem, it is straightforward to compute the smallest number of required training samples d_k, i.e., $d_k{=}\lceil\frac{D_k}{h+1}\rceil$. For the sake of simplicity, we assume that each node $n{\in}\mathcal{N}$ owns a dataset $\mathcal{D}_n$ such that $|\mathcal{D}_n|{\geq}d_k$, with $k = 1, \ldots, K$.

A requester will then ask $H{\geq}h$ other nodes in the cluster for help: indicating with $l{\leq}H$ the number of nodes that accept the request (hereinafter also referred to as *learners*), if $l{<}h$, the learning process fails and the next requester has to wait a time T_w before starting generating a new request. When instead at least h nodes agree to help, i.e., $h{\leq}l{\leq}H$, the training session can start: the set of l learners that accept the request and, thus, contribute to the training, remains the same till the training of the model is completed.

A training session is said to belong to type (k, j) when the requester belongs to class k and the minimum class of the set of the H nodes receiving the request is equal to j. The probability that a node accepts to cooperate in a training session of type (k, j) is denoted by π_{kj}. The dependence of acceptance probabilities upon the session type allows us to reproduce the reasoning of potential learners, which will also consider the expected amount of resources to devote to the process when deciding whether to accept.

When a training session begins, the requester sends the model to the l learners, which, in parallel and along with the requester, perform one learning epoch using their own local dataset. Nodes then exchange the updated models with each other, and combine newly-received models with their local one. A model training is terminated after a number of epochs e_k needed to achieve the target learning quality. Notice

that such a number depends on the requester class k, hence, on the complexity m_k of the model to train, and can be determined *a priori* by using an approach similar to the one in [140] if the model is already characterized. Alternatively, it is possible to use methods that find convergence bounds, such as the one proposed in [35]. The time required to perform one epoch is equal to the processing time taken by the slowest learner; in general, the local processing time of a node depends on the number of processed samples, its computing capability, and the model complexity m_k. For the sake of simplicity, we do not take into account the CPU consumption due to communication, i.e., to the radio functions enabling the transmission of the models' parameters.

To properly describe and assess the performance of the learning process, we define the following metrics:

- $x_n^{kj}(t)$: number of requests generated by node n for a training session of type (k, j) accepted till time t;
- $X_n^{kj}(t)$: number of requests generated by node n for a training session of type (k, j) till time t;
- $y_n^{kj}(t)$: number of requests made to node n for a training session of type (k, j) accepted till time t;
- $Y_n^{kj}(t)$: number of requests made to node n for a training session of type (k, j) till time t;
- $\mu_n^{kj}(t)=\frac{x_n^{kj}(t)}{X_n^{kj}(t)}$, $\nu_n^{kj}(t)=\frac{y_n^{kj}(t)}{Y_n^{kj}(t)}$, which indicate the quantity of service, (resp.) received and given by n for type-(k, j) training sessions till time t.

Last, we define the Normalized Received Service of a generic requester n for training sessions of type (k, j) as $\mathrm{NRS}_n^{kj}(t)=\lim_{t\to\infty}\mu_n^{kj}(t)$. Beyond its role in the mathematical formulation of the problem described below, this metric is very important to assess how willing nodes are to cooperate, hence, intuitively, how effective our cooperation scheme is. For simplicity, hereinafter, the NRS indices will be omitted.

4.3 Problem Formulation

As nodes are rational, each node aims to maximize its own long-term utility, that is its average *payoff* over all training sessions. Assuming that a node receives a payoff of 1 when its training request is successful, we define the utility of a node n of class k in a type (k, j) session as $U_n^{kj} = \lim_{t\to\infty} \frac{x_n^{kj}(t)}{S(t)} = P_n^{kj}\Pi_{kj}$, where $S(t)$ indicates the total number of sessions (accepted and rejected) till time t, P_n^{kj} denotes the probability that node n is a requester in a type (k, j) session, while Π_{kj} is the probability for a training session of type (k, j) to be accepted. The latter event occurs when the number l of accepting nodes is higher than the required minimum number h, i.e.,

$$\Pi_{kj} = P(l \geq h) = \sum_{\ell=h}^{H} \frac{H!}{\ell!(H-\ell)!} \pi_{kj}^{\ell}(1-\pi_{kj})^{H-\ell}, \tag{4.1}$$

where, as mentioned, we consider that all nodes accept to contribute to a training session of type (k, j) with the same probability π_{kj}. We will thus highlight below the dependency of the node's utility on π_{kj} by writing $U_n^{kj}(\pi_{kj})$.

Finally, denoting the mean computing expenditure of the generic node n with c_n, the objective of node n, belonging to class k and acting as a requester, is given by:

$$\max_{\pi_{kj}} \sum_{j} U_n^{kj}(\pi_{kj}), \quad \text{s.t. } c_n \leq \bar{\chi}_k, \forall n \in \mathcal{N}. \tag{4.2}$$

In the objective function above, the decision variables are the probabilities π_{kj} with which a node accepts to cooperate in a session of type (k, j). Note that, since all nodes are rational and are provided by the orchestrator with the same system information (number of nodes and their class), they will all end up computing the same optimal values for the π_{kj}'s.

4.3.1 Stationary case: closed-form solution

Given the problem above, we now assume stationary conditions in which all nodes consistently use the same π_{kj} values. This assumption, which will be removed in Sec. 4.4, is essential at this stage as it allows us to analytically derive the π_{kj}'s, as

follows. Let us write the mean duration of a session, T_s, as:

$$T_s = \sum_k \sum_j p_{kj} \Big[T_{kj} \Pi_{kj} + T_w (1 - \Pi_{kj}) \Big] \tag{4.3}$$

where the first and second terms on the right-hand side account for the case when a session is accepted and rejected, respectively. In particular, p_{kj} is the probability that an occurring session is of type (k, j), while T_{kj} is the training time for a session of type (k, j), i.e., $T_{kj} = \frac{m_k e_k d_k}{\chi^{\max}_{\min(k,j)}}$. In this latter expression, the term $m_k e_k d_k$ is the total number of FLOPs required to finish training the requester's model, while $\chi^{\max}_{\min(k,j)}$ is the smallest across the maximum computational capability of the nodes participating in the training session. The time required for transmitting and aggregating the model parameters is assumed to be negligible when compared to the computational time. Being T_s the mean session duration, it must hold: $T_s{>}0$.

As a node cannot be simultaneously a requester and a learner, we model c_n as a sum weighted on the probabilities that such events occur. The mean computing expenditure for a node n of class k when the node is a requester and a learner is (resp.) given by:

$$c_n^r = \sum_{j=1}^{K} P_n^{kj} \frac{m_k e_k d_k}{T_s} \Pi_{kj} \quad \text{and} \tag{4.4}$$

$$c_n^l = \sum_{j=1}^{K} \sum_{i=1}^{k} q_{ji}^{(k)} \frac{m_j e_j d_j}{T_s} \pi_{ji} \Omega_{ji}, \quad \text{where} \tag{4.5}$$

- $q_{ji}^{(k)}$ the probability for a node of class k to receive a training request in a session of type (j, i);
- Ω_{ji} denotes the probability that out of the remaining $H{-}1$ nodes that receive the request at least $h{-}1$ accept.

It follows that $c_n{=}c_n^r + c_n^l$, where the weights of the sum are inherently contained in c_n^r and c_n^l. By considering the constraint of the node's computing capability in (4.2), we get K inequalities and K^2 unknown variables. To solve the system, we need to reduce the number of variables to K, which can be done by exploiting the nodes' *rationality*, as described below.

Rationality. Consider a system with only $N{=}2$ nodes belonging to the same class, i.e., $K{=}1$ and $N_1{=}N$. By rationality, each node must have the same NRS

value. Indeed, having the same average computing capability constraint, one node cannot increase its NRS without reducing the NRS of the other node; the latter, however, would not find such an operating point acceptable. Therefore, the NRSs must be equal for both nodes. This can be extended to the case where all $N>2$ nodes belong to the same class: also in this scenario, by rationality, each node must have the same value of NRS. In this case, it is straightforward to derive the value of π_{11} that maximizes the objective function while satisfying the computing capabilities constraint.

Next, consider a system with $N_1{=}n_1$, $N_2{=}1$, $H{=}h{=}1$, and that the quantity of received service is the same for all requesters when the learner belongs to class 1, i.e.,

$$m_1 e_1 d_1 \Pi_{11} = m_2 e_2 d_2 \Pi_{21} \,. \tag{4.6}$$

Further, by rationality, the quantity of service received by class 1 nodes from the node in class 2 must be equal to the quantity of service received by the latter from the nodes in class 1. Indeed, the class-2 node has no interest in being more generous to the nodes of class 1 than what the latter ones are to the class-2 node, since the latter will not receive a higher service share anyway. Thus, we have:

$$m_1 e_1 d_1 \;\; \Pi_{12} = m_2 e_2 d_2 \;\; \Pi_{21} \,. \tag{4.7}$$

From (4.6) and (4.7), it derives that: $\Pi_{11}{=}\Pi_{12}$. Notably, in this simple scenario where $H{=}h{=}1$, from (4.1) one can derive that $\Pi_{kj}{=}\pi_{kj}$ for all possible session types (k, j).

We now extend (4.6) and (4.7) to the most general case, i.e., with $K{\geq}1$, $H{\geq}1$ and $h{\geq}1$. In this case, we have:

(i) the values of Π_{kj} are such that $\Pi_{kk}{=}\Pi_{kj}, 1{\leq}k{\leq}j{\leq}K$, which implies

$$\pi_{kk} = \pi_{kj}, \quad 1 \leq k < j \leq K; \tag{4.8}$$

(ii) when a node receives a training request for a session of type (j, k), the rational values of Π_{jk} are such that

$$m_k e_k d_k \;\; \Pi_{kj} = m_j e_j d_j \;\; \Pi_{jk}, \quad 1 \leq k < j \leq K, \tag{4.9}$$

from which one can compute the ratio between π_{kj} and π_{jk}, $1 \leq k < j \leq K$.

Thus, given (4.8) and (4.9), and solving the aforementioned system of K inequalities by using (4.1) and (4.3)-(4.5), one can compute the π_{kj} values that maximize (4.2) and allow obtaining the Pareto optimal NRS values, i.e., values such that no node can improve its NRS without decreasing another node's NRS, while meeting the constraint on the mean computing expenditure. Importantly, (4.8) and (4.9) guarantee, in their respective cases, equal service shares received by all nodes; hence the Pareto optimal π_{kj} values are fair.

4.4 GENIAL: Algorithm and Properties

In the previous section, we have derived the π_{kj}'s under the stationary assumption, which implies that all nodes consistently use such values. However, since the nodes are rational, they cannot be relied upon to do so; intuitively, nodes will try to exploit their peers' naivety by first accepting the cooperation of other nodes, and then refusing to reciprocate. It follows that *no* stationary acceptance policy would be feasible; therefore, to effectively foster cooperation among nodes, a *behavioral* strategy is required, whereby nodes decide their actions based upon past history. Specifically, we propose a novel game-theoretic approach called GENerous Incentive Algorithm for Learning (GENIAL), which is based on the GTFT strategy and allows attaining the derived π_{kj}'s.

Consider a node n receiving at time t a request from a node of class k, that the smallest class among the H nodes that have received the request is j; thus the learning session is of type (k, j). To accept or deny the request, n applies the GENIAL algorithm, that is:

- if $\nu_n^{kj}(t) > \pi_{kj}\Omega_{kj}$ or $\mu_n^{ik}(t) < \frac{\Pi_{ik}}{\pi_{kj}\Omega_{kj}}\nu_n^{kj}(t) - \gamma$ reject;
- else accept.

According to this strategy, n rejects the requests if at least one of the following two conditions holds:

- $\nu_n^{kj}(t) > \pi_{kj}\Omega_{kj}$, i.e., in sessions of type (k, j) node n has performed more training than what it should have;

- $\mu_n^{ik}(t) < \frac{\Pi_{ik}}{\pi_{kj}\Omega_{kj}} \nu_n^{kj}(t) - \gamma$, with $\gamma > 0$, i.e., the amount of help received by node n in type-(i,k) sessions is lower than the amount of help it has given in type-(k,j) sessions, normalized by the term $\frac{\Pi_{ik}}{\pi_{kj}\Omega_{kj}}$, which is the value to which the ratio $\frac{\mu_n^{ik}(t)}{\nu_n^{kj}(t)}$ should converge. Since γ is a small positive value, nodes are a little generous by agreeing to train for others even if they have not received as much help as they should have.

We now prove that GENIAL yields a Nash Equilibrium, first in the simple case where all nodes belong to the same class and only one learner is necessary for training (i.e., $K=H=h=1$).

Theorem 1. *Consider a system of N nodes, all belonging to class 1, with mean and maximum computing capability constraint (resp.) $\bar{\chi}_1$ and $\chi_1^{\max}$. Models have complexity m_1, which implies that d_1 samples and e_1 epochs are required to reach the target learning quality. Also, $H=h=1$, and if the node receiving the training request accepts, the time required for the training is T_{11}, otherwise a time T_w is waited. Then:*

1. *if all nodes, except for node n, employ GENIAL, then* $\limsup_{t\to\infty} \mu_n^{11}(t) \leq \pi_{11} = \frac{\bar{\chi}_1 N}{2} \frac{T_w}{m_1 d_1 e_1 - \frac{\bar{\chi}_1 N}{2}(T_{11} - T_w)}$;
2. *if all nodes apply GENIAL, then* $\lim_{t\to\infty} \mu_n^{11}(t) = \pi_{11} = \frac{\bar{\chi}_1 N}{2} \frac{T_w}{m_1 d_1 e_1 - \frac{\bar{\chi}_1 N}{2}(T_{11} - T_w)}$, $\forall n \in \mathcal{N}$.

Proof. Please see the Appendix A. □

Next, we move to the multi-class case.

Theorem 2. *Consider a system of N nodes and K classes, $H=h=1$, and N_k nodes in class k. Also, the computing capabilities constraints are as follows: $\bar{\chi}_1 < \bar{\chi}_2 < \ldots < \bar{\chi}_K$ and $\chi_1^{\max} < \chi_2^{\max} < \ldots < \chi_K^{\max}$. Then:*

1. *if all nodes, except for node n, employ GENIAL, then* $\limsup_{t\to\infty} \mu_n^{kj}(t) \leq \pi_{kj}$;
2. *if all nodes apply GENIAL, then* $\lim_{t\to\infty} \mu_n^{kj}(t) = \pi_{kj}$, $\forall n \in \mathcal{N}$ *and* $k, j = 1, \ldots, K$.

Proof. Please see the Appendix A. □

Table 4.1 Small-scale scenario: additional settings

	Class 1	Class 2	Class 3
Dataset size	3×10^3	5×10^3	7×10^3
Model complexity [FLOPs]	2×10^3	3×10^3	4×10^3
Epochs required	40	55	70
$\chi^{\max}$ [MFLOPS]	45	60	75
$\bar{\chi}$ [MFLOPS]	8.4	12	15

From Theorems 2 and 3, it is easy to show, by using randomizing arguments, that GENIAL constitutes a Nash Equilibrium and allows converging to the fair Pareto optimal operating point also when considering more than one node receiving a training request ($H>1$), and more than one learner ($h>1$).

Theorem 3. *Consider a system with N nodes, K classes, $H>1$, $h>1$ and N_k nodes in class k, $k=1,\ldots,K$. As for the computing capabilities constraints, assume that $\bar{\chi}_1<\bar{\chi}_2<\ldots<\bar{\chi}_K$ and $\chi_1^{\max}<\chi_2^{\max}<\ldots<\chi_K^{\max}$. Then:*

1. *if all nodes, except for node n, use GENIAL, $\limsup_{t\to\infty}\mu_n^{kj}(t)\leq\Pi_{kj}$;*
2. *if all nodes apply GENIAL, then $\lim_{t\to\infty}\mu_n^{kj}(t)=\Pi_{kj}$, $\forall n\in\mathcal{N}$ and $k,j=1,\ldots,K$.*

Proof. Please see the Appendix A. □

4.5 Numerical Results

In this section, we experimentally analyze the behavior of the nodes when applying GENIAL, as well as the effects of GENIAL on the social utility of the system.

Small-scale scenario. We consider a set of $N=12$ nodes and $K=3$ classes, with four nodes in each class training the models in Tab. 4.1. Also, we set $H=h=2$, i.e., the requester asks two nodes for help and both need to accept for the training to take place. All nodes employ GENIAL, with T_w set to 1 s.

The Π_{kj} values (hence, the Pareto optimal NRSs) are obtained by solving the equations (4.3)–(4.5) jointly with (4.8), (4.9) and the K inequalities stating the computing capability constraints. Fig. 4.1 depicts such values (black markers), along

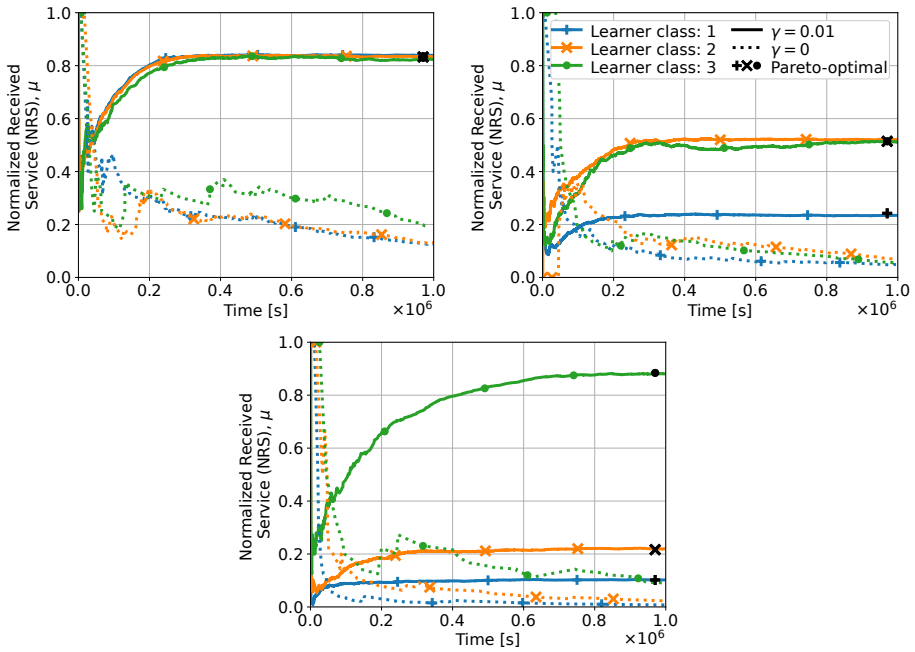


Fig. 4.1 Small-scale scenario: Time evolution of the NRS of a requester in class 1 (left), 2 (center), and 3 (right), with the fair Pareto optimal values indicated by the black markers. NRSs converge to the target value only for γ=0.01 (solid lines), while for γ=0 (dotted lines), they converge to much lower values.

with the temporal evolution of the requesters' NRS. Specifically, the plots refer to requesters belonging, respectively, to class 1 (left), 2 (center), and 3 (right). One can observe that, when γ=0.01 (solid lines), the NRS values converge to the fair Pareto optimal values, while they reach much smaller values for γ=0 (dotted lines). The results thus confirm that being moderately generous (γ=0.01) yields much better performance, even if the nodes are self interested.

We now consider the same scenario, but with one and two nodes per class, hence, a total of (resp.) 3 and 6 nodes, that use $\gamma = -0.02$ and, hence, are slightly selfish. Hereinafter we refer to such nodes as *parasites*. Fig. 4.2 presents the NRS values for each session type in such scenarios, and compares them to the case without parasite nodes. Notice how, with 3 parasite nodes out of 12 (i.e., 25% parasites), the NRSs are lower than in the case with all nodes being generous, and, with 6 parasites (i.e., 50% parasites), the performance degrades to such an extent that the NRSs approach 0. Importantly, this outcome underlines that, by using GENIAL, all nodes get lower NRSs if a selfish (i.e., non-collaborative) behavior is adopted; as a consequence, rational nodes have no incentive to follow an ungenerous behavior.

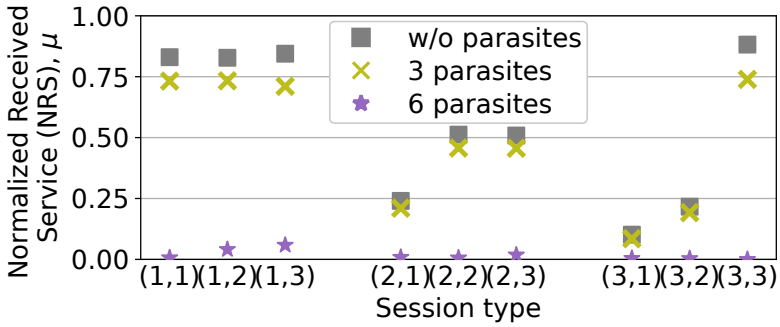


Fig. 4.2 Small-scale scenario: NRS values for each session type in three different cases: without parasites, and with 3 (25%) and 6 (50%) parasites that set $\gamma = -0.02$. The remaining nodes have γ=0.01.

Fig. 4.3 sheds light on the resource usage for generous and parasite nodes. When no parasites are present (solid, grey bars), the nodes fully exploit the available resources, hence, the resource-usage constraint is met with an equal sign. This is what we can expect from a system that works well, adequately exploiting the available resources. On the contrary, in the scenario with 3 parasites (green), parasite nodes consume a slightly smaller amount of resources (checked bars), but, as shown by Fig. 4.2, at a cost of a lower NRS. With 6 parasites (purple), the computational expenditure drops dramatically for all nodes, as, essentially, very few models get trained.

The NRS values are an indicator of individual utility, as they refer to each single node. To analyze the social utility, i.e., the benefit for the set of nodes as a whole, we show in Tab. 4.2 the normalized number of accepted, and, hence, carried out, training sessions. The obtained social utility values are consistent with the results presented above: the scenario where all nodes are generous (all use γ=0.01) yields the larger social utility, confirming the importance of being generous also from the social utility point of view.

In conclusion, these results emphasize that to foster cooperation in a decentralized learning task and reach the Pareto optimal NRS values, all nodes must employ GENIAL with γ>0: as nodes are rational, they are motivated to be generous.

Larger-scale scenario. We now consider K=3 and N_1=12, N_2=18, and N_3=30, for a total of 60 nodes, and we set H=5, h=4, and T_w=1 s. All nodes employ GENIAL with γ=0.01; the additional settings for this scenario are reported in Tab. 4.3. Tab. 4.4 presents the obtained NRSs, which converge to the fair Pareto

Table 4.2 Social utility in different scenarios

Scenario	Social utility
w/o parasites, γ=0.01	0.454
w/o parasites, γ=0	0.011
25% parasites	0.398
50% parasites	0.014

optimal values, shown in parentheses. This is consistent with the small-scale scenario and highlights how GENIAL can work effectively even when the number of nodes grows.

4.6 Related work

Our study leverages game theory to foster cooperation among the nodes participating in decentralized learning tasks. We draw on [145], which considers a wireless ad-hoc network where source nodes can communicate with their target destination only if intermediate nodes accept to relay the packets to be delivered. Here, we extended the game-theoretic framework in [145] to make it suitable for decentralized learning scenarios, where nodes are asked to collaborate, by making available their computational resources to train ML models.

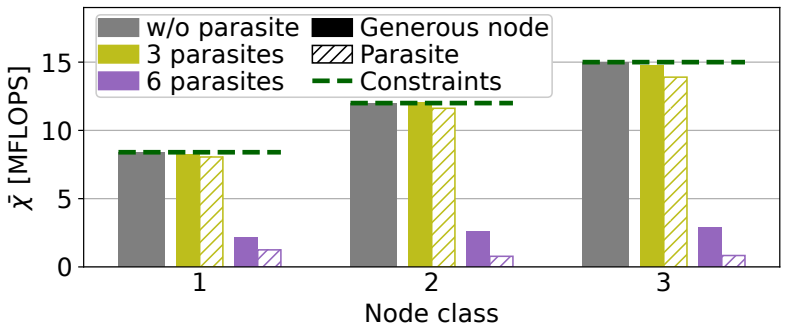


Fig. 4.3 Small-scale scenario: $\bar{\chi}$ for each class of nodes in three different cases: without parasites, and with 3 (25%) and 6 (50%) parasites that set $\gamma = -0.02$. The remaining nodes have γ=0.01. Dashed lines depict the $\bar{\chi}$ constraints for the respective classes of nodes.

Table 4.3 Larger-scale scenario: additional settings

	Class 1	Class 2	Class 3
Dataset size	2×10^3	2.5×10^3	3×10^3
Model complexity [FLOPs]	1×10^3	2×10^3	2.5×10^3
Epochs required	35	40	45
$\chi^{\max}$ [MFLOPS]	30	45	54
$\bar{\chi}$ [MFLOPS]	15.1	24.9	26.9

Table 4.4 Larger-scale scenario: NRSs at convergence, with Pareto optimal values in parentheses

		Requester's class		
		1	2	3
Learner's class	1	0.63 (0.63)	0.22 (0.22)	0.13 (0.13)
	2	0.63 (0.63)	0.67 (0.67)	0.40 (0.40)
	3	0.63 (0.63)	0.67 (0.67)	0.97 (0.99)

As for decentralized learning, [142] presents a comprehensive theoretical analysis of convergence in decentralized settings and introduces a unified framework for Decentralized Stochastic Gradient Descent (SGD) optimization, which generalizes earlier methods and is proven to achieve the optimal convergence rates for local SGD across different scenarios. [141] theoretically and experimentally demonstrates the convergence speedup of a decentralized optimization algorithm on networks with low bandwidth or high latency, when compared to a distributed scenario with a central node aggregating the updates computed by the training nodes. The latter case is generally referred to as distributed learning, whose most popular representative is Federated Learning (FL) [7]. Importantly, also FL, when deployed in real scenarios, requires nodes to be encouraged to train the model by means of incentive mechanisms [147], which can be based on economic or game-theoretic approaches [148].

The authors of [149] consider a distributed learning scenario with rational nodes and a Fusion Center, i.e., the central coordinator, that receives the gradient updates from the nodes and fosters their cooperation by using a reward mechanism based on zero-determinant strategies. Such interactions are treated as repeated games.

Similarly, [150] models the interactions between the FL coordinator and the clients by using a Stackelberg game.

The study in [151] introduces an incentive mechanism for cross-silo FL, specifically designed to account for participating organizations' heterogeneity and the concept of public goods. This mechanism achieves social welfare maximization, while ensuring individual rationality and budget balance. [152] models FL as a hedonic game, where players form coalitions based on a cost, given by the weighted sum of the players' errors. Therein the authors propose an algorithm to find an optimal arrangement of players and analyze the trade-off between stability and optimality using the Price of Anarchy.

4.7 Conclusions

We considered a set of rational nodes that can cooperatively train or fine-tune ML models according to the decentralized learning paradigm. To incentivize the nodes to participate in the training, we envisioned a game-theoretic mechanism, GENIAL, that achieves the nodes' fair Pareto optimal operating points and constitutes a Nash equilibrium. Our numerical experiments support the theoretical findings, and indicate that the presence of parasite nodes seeking to improve their short-term benefit by deviating from the Nash equilibrium hurts the performance of *both* generous and parasite, thus discouraging a parasite behavior.

This chapter concludes the first part of this thesis, where we focused on two main research areas of Cooperative Learning, specifically the optimization of energy consumption entailed by the training of ML models and the development of incentive mechanisms to promote collaboration among learning nodes.

In the following part of this manuscript, we will introduce the diffusion models, which have emerged as a powerful class of generative models for their remarkable capability to produce high-quality synthetic data. Specifically, we will consider a multi-view image scenario, where multiple novel views of a given object must be generated. First, we will develop a diffusion-based Novel View Synthesis model that achieves state-of-the-art performance; then, we will show the benefits of employing such a model to improve the performance and robustness of a classifier.

Chapter 5

DiMViS: Diffusion-based Multi-View Synthesis

Multi-view observations offer a broader perception of the real world, compared to observations acquired from a single viewpoint. While existing multi-view 2D diffusion models for novel view synthesis typically rely on a single conditioning reference image, a limited number of methods accommodate a multiple number thereof, by explicitly conditioning the generation process through tailored attention mechanisms. In contrast, we introduce DiMViS, a novel method enabling the conditional generation in multi-view settings by means of a joint diffusion model. DiMViS capitalizes on a pre-trained diffusion model, while combining an innovative masked diffusion process to implicitly learn the underlying conditional data distribution, which endows our method with the ability to produce multiple images given a flexible number of reference views. Our experimental evaluation demonstrates DiMViS's superior performance compared to current state-of-the-art methods, while achieving reference-to-target and target-to-target visual consistency.

Part of the work described in this chapter has been already published in **Di Giacomo, G.**, Franzese, G., Cerquitelli, T., Chiasserini, C. F., Michiardi, P. (2024). DiMViS: Diffusion-based multi-view synthesis. In ICML 2024 Workshop on Structured Probabilistic Inference & Generative Modeling.

5.1 Introduction

Generative models play a pivotal role in learning data distributions, which endows them with the capability of generating synthetic samples that closely resemble real-world data, such as, for example, images or text. Over the years, Generative Adversarial Networks (GANs) [153] and Variational AutoEncoders (VAEs) [154] have dominated the field; however, recently, diffusion models [155, 156] have emerged as a powerful alternative. Specifically, Latent Diffusion Models (LDMs), such as Stable Diffusion [157], have become the state-of-the-art for image generation: large-scale training on billions of 2D images [158] equips such models with rich semantic priors and, hence, a strong generalization capability.

Motivated by the impressive performance of diffusion models, many works trained 3D variants thereof; despite their remarkable results, 3D datasets are still inadequate compared to the large available 2D datasets. To fill this gap, Zero123 [159] capitalizes on the rich 2D priors provided by Stable Diffusion, leveraging multi-view image data to learn 3D priors: once trained, given a single-view image, Zero123 is able to generate images of the underlying object from different target viewpoints. Subsequent works build on Zero123 to improve the generated views consistency [160, 161].

Such methods perform the task usually referred to as Novel View Synthesis (NVS) and try to emulate the experience-based human capability of inferring 3D shapes and occluded views from a *single* observed reference view. However, in real-world scenarios, observation may be acquired from different viewpoints, offering a more comprehensive understanding compared to single-view observations, which may fall short in capturing the complexity and diversity of an object or a scene. Driven by this motivation, [162] extend Zero123, to allow multiple conditioning views for generating a novel image from a target viewpoint.

We provide a detailed review of the related work in 5.2, where we underline that most existing approaches lack flexibility in accommodating a varying number of both input and output views, namely reference and generated images. This drawback hinders such methods' deployment in practical applications requiring higher flexibility. Consider a set of recording cameras used, for instance, to ensure people's safety or to improve a classification task: in these real-use case scenarios, cameras may malfunction or be occluded. In such circumstances, we are interested in recovering

the missing views, considering the observed ones. Consequently, the number of input views and images that must be generated cannot be fixed *a priori*.

To this end, [163, 164] propose a diffusion model based on a multi-branch U-Net, with branches sharing the same architecture and weights; a similar approach is also envisioned in [165]. Importantly, these methods employ elaborate attention mechanisms to ensure consistency among conditioning and target views and across the target views themselves. Differently from this approach, we propose a novel method called **Diffusion-based Multi-View Synthesis (DiMViS)** that we specifically design for *conditional generation by implementing a mechanism that allows latent variables to evolve according to different time values in the forward process, and that produces a correlation between latent variables in the backward process through a joint diffusion model*.

At inference time, given a fixed-size set of images, DiMViS accepts any number of reference views to conditionally generate the missing ones. DiMViS uses the pre-trained weights of Stable Diffusion Image Variations [166], while adopting both for training and inference the masked diffusion method introduced in [167], which addresses the problem of modeling multiple input modalities (image, audio, and text data) representing the same concept. Remarkably, DiMViS does not need any addition or change with respect to the standard attention mechanism employed in the pre-trained model, as it relies on a joint diffusion model that allows latent image variables to mutually influence one another. Therefore, DiMViS is able to effectively aggregate multiple reference views: when incorporating additional images, thanks to the larger conditioning information, DiMViS performance is enhanced while the inherent stochasticity of the image generation process is reduced. We experimentally find that DiMViS outperforms current state-of-the-art NVS 2D diffusion models while achieving visual consistency of the generated images.

To summarize, DiMViS addresses the following key requirements:

- **Generalization** capability inherited by the strong 2D generative prior provided by Stable Diffusion;
- **Flexibility**, as it both accepts and generates multiple views by leveraging a joint diffusion model that seamlessly combines masked diffusion with the pre-trained model;

- **Efficient multi-view aggregation**, with improvement up to 15% in perceptual similarity when multiple views are available.

5.2 Related work

In this section, we divide the current novel view synthesis approaches into two main categories, namely, Neural Radiance Fields (NeRFs) and 2D diffusion models.

5.2.1 NeRF models

NeRFs [168] are powerful methods that pioneered a substantial advancement in 3D scene reconstruction and rendering, enabling NVS. While the first models required a per-scene optimization using dense input views and accurate camera poses, recent works have tried to overcome these limitations. BARF [169] trains a NeRF from noisy or unknown camera poses, but assumes dense input views. FORGE [170] is a NeRF variant that jointly predicts the input images' camera poses, which are assumed to be unknown, and performs the object reconstruction. FORGE is endowed with generalization capability and carries out a quick test-time optimization for further performance improvement; nevertheless, it requires at least 2 input images to work. SPARF [171] is a NeRF variant that performs effectively given only a few sparse input images, i.e., only a few views are available, with noisy camera poses, which are jointly optimized together with the radiance field; however, it requires per-scene optimization.

To deal with the challenges related to NeRF models, LEAP [172] proposes a generalizable NeRF that can model 3D scenes or objects given a sparse set of views, relying only on their relative camera poses. However, [164] highlights that its proposed diffusion-based method achieves much better image quality than LEAP: according to the authors, the reason relies upon the strong image priors inherited from the pre-trained latent diffusion models. Furthermore, experimental results in [165] demonstrate the superior generation quality of a reference NeRF model [173] with respect to their diffusion-based method only when considering a dense set of conditioning views; nonetheless, the NeRF model requires scene-specific training and is outperformed in sparse-view settings. In such a scenario, we argue that novel view synthesis should be modeled as a conditional generation problem: when a part of an

object is not visible, it can be only obtained by using a generative model, such as a diffusion model, and not NeRF approaches, which typically lack generative capability as they do not rely on generative modeling. By doing so, it is possible to take into account the stochasticity of the generation process and, hence, to produce different plausible images. Importantly, the generation randomness should decrease when increasing the number of reference views.

In this direction, [174] design a category-specific model for 3D reconstruction that leverages a diffusion model to optimize a NeRF, through a diffusion distillation mechanism. A similar approach is proposed in [175], where a NeRF is coupled with a diffusion model conditioned on the CLIP embeddings of the available views and a feature map containing their geometric information. However, such proposed models are only tested when more than one input view is available, whereas in real-use case scenarios only a single image may be observed.

5.2.2 2D Diffusion models

The remarkable generative capability of 2D diffusion models [157] has motivated the development of NVS methods built upon them. 3DiM [176] uses a diffusion model operating in the image space and allows more than one input image to condition the novel view synthesis; however, it generates only one output image, whose quality, counterintuitively, becomes worse when increasing the number of conditioning inputs.

Zero123 [159] paved the way to 2D diffusion models for NVS thanks to its strong generalization capability inherited by Stable Diffusion; however, it accepts only one conditioning image and generates a single output; thus, when creating images of the same object from multiple target viewpoints, this approach suffers from inconsistency as the images are generated independently. To overcome this issue, [177, 161] extend Zero123 in order to generate multiple consistent views simultaneously, while [160] propose a diffusion model, still based on Zero123, that uses a 3D-aware feature attention mechanism to ensure consistency across the produced images. To achieve this goal, [178] use epipolar attention for geometric awareness and multi-view attention, which better combines information from multi-view images. The latter method generates different images simultaneously and can potentially be fed with a sparse set of multi-view images, but all experiments are performed with only one reference image. Nevertheless, using only a single conditioning input may

be a limitation in real use-case applications: to address this drawback, [162] modifie the Zero123 architecture to accommodate multiple views in input, but still generates a single image from a target viewpoint. [164] propose a pose-free method to generate at the same time multiple images given a sparse set of views. To do so, the model employs a U-Net with several branches, i.e., copies, whose number is equal to the number of total views, both conditioning and target. Notably, a global self-attention mechanism among the U-Net features across all branches is applied to enhance the consistency of the generated views. A similar methodology based on a multi-branch U-Net is envisioned in [163]. [165] employ transformers [179] that, by leveraging specifically designed camera positional encodings, aim to improve reference-to-target and target-to-target views consistency.

Another area of research focuses on 3D-aware diffusion models, combining both diffusion models and NeRF-like 3D representations. [180] present a diffusion model that integrates geometric priors by conditioning it on 3D features. The model takes a variable number of input views, while still generating a single output image. Also, the model is category-specific. [181] introduce a diffusion model based on a multi-view U-Net that produces a neural scene representation, instead of images, by leveraging cross-view attention to align the induced features, similar to the previously mentioned works. Notably, the model is trained from scratch on a limited set of classes of different datasets. This hinders the comparison with our method, which exhibits generalization capability thanks to the 2D prior inherited by the pre-trained Stable Diffusion model.

Novelty. We remark that, conversely to the works based on explicit conditioning by means of tailored attention modules, DiMVis employs a joint diffusion model endowed with conditional generation ability by leveraging a straightforward yet effective masked diffusion mechanism that encourages correlation across the latent variables, which influence each other.

5.3 Preliminaries

For the sake of clarity, we summarize in this section the joint latent diffusion model introduced in [167], which we adapt to the NVS domain.

Given the set of input views $X=\{X^1,\dots,X^V\}$, first, a (deterministic) encoder e_ϕ produces latent variable Z^v for each X^v. We denote with $q_\phi(z)$ the produced distribution of the concatenated latent variable $Z=[Z^1,\dots,Z^V]$. A score-based diffusion model is then employed to learn $q_\phi(z)$, which endows the model with the capability to generate a new sample $\hat{Z}=[\hat{Z}^1,\dots,\hat{Z}^V]$. The score-based diffusion model relies on two stages, namely, the forward and the backward diffusion processes. The forward process is a stochastic noising process injecting noise into the input data, i.e., the latent representations, and is defined by the following Stochastic Differential Equation (SDE):

$$\mathrm{d}R_t = \alpha(t)R_t\mathrm{d}t + g(t)\mathrm{d}W_t, \;\; R_0 = Z \sim q(r,0), \tag{5.1}$$

where $\alpha(t)R_t$ and $g(t)$ are the drift and diffusion terms, respectively. W_t is a Wiener process, while $q(r,t)$ denotes the time-varying probability density of the stochastic process at time $t\in[0,T]$, with finite T and initial conditions $q(r,0)=q_\phi(r)$.

To generate a new sample, we need to reverse the noising process by simulating the reverse-time SDE:

$$\mathrm{d}R_t = \Big(-\alpha(T-t)R_t + g^2(T-t)\nabla\log(q(R_t,T-t))\Big)\,\mathrm{d}t + g(T-t)\mathrm{d}W_t, \;\; R_0\sim q(r,T). \tag{5.2}$$

To solve (5.2), a parametric score network $s_\chi(r,t)$ is used to approximate the true score function; furthermore, $q(r,T)$ is approximated with the noise distribution $\epsilon\sim\mathcal{N}(0,I)$. Finally, a decoder d_ψ is used to map back the latent variables into the input space.

Notably, the encoder e_ϕ and the decoder d_ψ are the two building neural networks of what is usually referred to as autoencoder, a model designed to compress data and reconstruct it. Specifically, the encoder produces a latent variable that lies in a lower-dimensional space; then, the latent variable is fed into the decoder, which generates an output that should closely resemble the input.

Conditional generation. Our model accommodates conditional generation: specifically, the model leverages masked forward and backward diffusion processes to produce samples from the conditional distribution $q_\phi(z^M \mid z^C)$, being C and M the sets of conditioning and missing views to be generated, and, hence, z^C and z^M the

respective latent variables. Formally, we define the masked forward SDE as:

$$\mathrm{d}R_t = \mathcal{M}(M) \odot [\alpha(t) R_t \mathrm{d}t + g(t)\mathrm{d}W_t], \quad q(r,0) = q_\phi(r^M \mid z^C)\delta(r^C - z^C), \tag{5.3}$$

where $R_0{=}\mathcal{C}(R_0^M, R_0^C)$, with $R_0^M{\sim}q_\phi(r^M \mid z^C)$, $R_0^C{=}z^C$, and $\mathcal{C}(\cdot)$ being the concatenation operator. Importantly, the mask $\mathcal{M}(M)$ is used to freeze or diffuse the latent variable z^C and z^M, respectively.

The reverse-time process of (5.3) is defined as follows:

$$\begin{aligned}\mathrm{d}R_t{=}\mathcal{M}(M){\odot}\,[(-\alpha(T{-}t)R_t{+}\\ g^2(T{-}t)\nabla \log\Big(q(R_t, T{-}t \mid z^C)\Big)\Big)\,\mathrm{d}t{+}g(T{-}t)\mathrm{d}W_t\Big],\end{aligned} \tag{5.4}$$

with $R_0{=}\mathcal{C}(R_0^M, z^C)$ and $R_0^M{\sim}q(r^M, T \mid z^C)$. Also in this case, $q(r^M, T \mid z^C)$ is approximated by its corresponding steady state distribution $\epsilon{\sim}\mathcal{N}(0, I)$, and the true conditional score function $\nabla \log\Big(q(r, t \mid z^C)\Big)$ is estimated with a conditional score network $s_\chi(r^M, t \mid z^C)$.

The joint diffusion model in [167] implements masked diffusion by using a multi *multi-time vector* $\tau{=}[t_1, \ldots, t_V]$, which concurrently indicates the diffusion time and which views are missing. Formally, the multi-time vector is defined as $\tau(M,t){=}t\left[\mathbb{1}(1{\in}M), \ldots, \mathbb{1}(V{\in}M)\right]$. Note that this formulation can be easily adapted to include conditioning signals, such as CLIP [182] embeddings.

Finally, the original continuous-time model definition from [167] can be cast in discrete-time, such as Denoising Diffusion Probabilistic Model (DDPM) [155]. Indeed, continuous- and discrete-time diffusion converge to a mathematically equivalent expression with a sufficiently high number of integration steps. In DiMViS we use a discrete-time diffusion formulation, which enables our approach to exploit pre-trained models such as Stable Diffusion [157].

5.4 Our Method: DiMViS

Given a set of V views, we define the generic subsets of conditioning, i.e., reference, and missing views as $X^C{=}\{X^c\}_{c\in C}$ and $X^M{=}\{X^m\}_{m\in M}$, where C and M are the sets of indices of conditioning and missing views, respectively. The goal is to generate

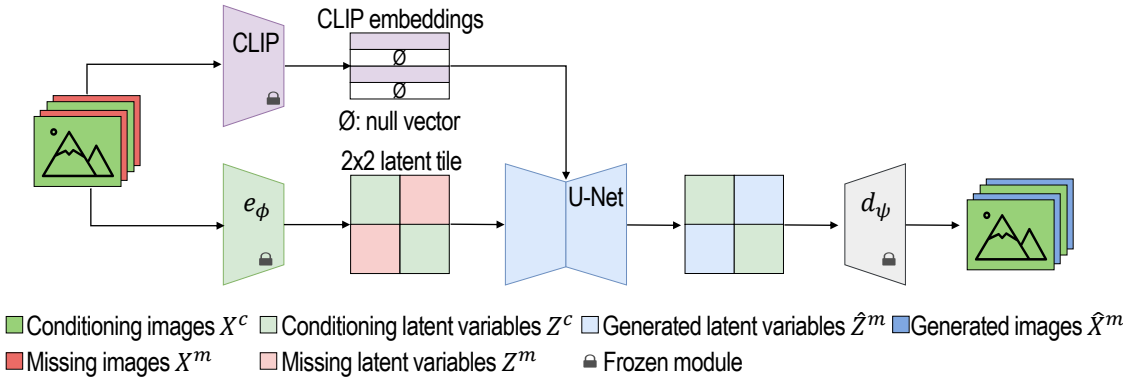


Fig. 5.1 **DiMViS Architecture**. Given a set of multi-view images, DiMViS is able to generate the missing views given the conditioning observed views. In this example, we consider two conditioning images and two missing ones.

the missing views given the reference ones: we model this problem as a conditional generation task, by using a *joint* diffusion model.

To exploit the generalization ability provided by the strong 2D prior acquired through large-scale training of pre-trained 2D diffusion models, we capitalize on the Stable Diffusion [157] architecture; specifically, we initialize the model weights from Stable Diffusion Image Variations v2 [166], which was obtained by fine-tuning Stable Diffusion v1.4. However, we need to accommodate a variable number of reference and target views at inference; consequently, to learn the conditional data distribution, we adapt the original architecture to fine-tune it according to the Masked Multi-time diffusion approach introduced in [167] for the multi-modal domain.

At inference, our approach is depicted in Fig. 5.1, where we consider a set of $V{=}4$ views in total and, for example, the set of missing views with $M{=}\{2,4\}$. A deterministic encoder e_ϕ is used to encode each conditioning reference view X^c, with $C{=}\{1,3\}$, to obtain their latent representations $Z^c{=}e_\phi(X^c)$; on the other hand, the missing views latent variables are represented using random noise, sampling from the Normal distribution $\mathcal{N}(0,I)$. Such latent representations are concatenated along the height and width axes, forming a 2×2 tile of latent variables, which is the input of the U-Net.

Our architecture uses a conditioning branch containing a CLIP image encoder. The CLIP embeddings obtained from the reference views are concatenated together with the null vectors corresponding to the missing views embeddings and injected into

the cross-attention modules of the U-Net. The resulting concatenated embeddings vector has two purposes: it both contains the conditioning signals relative to the reference views and, thanks to the null vectors, indicates which views are missing. In [167], the information concerning the missing modalities is provided by a multi-time vector; however, by using the CLIP embeddings, we integrate such conditioning signal without any modification to the U-Net, which instead would be necessary with the multi-time vector, which can interfere with the pre-trained model.

Finally, the U-Net generates the latent variables $\hat{Z}^m$ of the missing views X^m, and a deterministic decoder d_ψ transforms the generated latent variables back into the input space, obtaining the generated images $\hat{X}^m{=}d_\psi(\hat{Z}^m)$.

5.4.1 Training procedure

During training, we freeze the pre-trained encoder e_ϕ, the decoder d_ψ and the CLIP image encoder, and we fine-tune the whole U-Net, to learn the conditional distribution of the missing latent variables $Z^M{=}\{Z^m\}_{m\in M}{\sim}p(z^M|z^C)$. To do so, we employ the masked diffusion approach introduced in [167], by using a complete training set and randomly setting some views as missing during the fine-tuning. Specifically, with probability $d{=}0.2$ we set $C{=}\emptyset$, i.e., there is no conditioning view, hence, we diffuse all latent variables and all the CLIP embeddings are null-vectors; on the other hand, with probability $1{-}d$, we perform masked diffusion: first, we uniformly sample the set of conditioning views over all the possible sets; then, the remaining views, which are assumed to be missing, are diffused and their CLIP embedding is set to the null-vector, while the latent variables of the conditioning views are frozen.

Formally, we fine-tune the denoiser U-Net ϵ_θ by minimizing the following loss:

$$\mathcal{L} = \lambda(M,C)\|\mathcal{M}(M)\odot[\epsilon-\epsilon_\theta(R_t,t,\mathrm{CLIP}(M,X))]\|_2^2, \tag{5.5}$$

where R_t is obtained by first sampling from the distribution $q(r\mid Z,t)$ and aggregating it with the input Z using the mask $\mathcal{M}(M)$. Importantly, we use a scaling factor $\lambda(M,C){=}1{+}\frac{|C|}{|M|}$ to take into account the randomization of M and C that leads to the diffusion of different portions of the latent space.

5.5 Experiments

5.5.1 Training details

We fine-tune DiMViS on Objaverse-1.0 [183], a large object-centric dataset containing more than 800k 3D models, which is commonly used for multi-view diffusion models. However, the original dataset contains many samples with poor quality, that may negatively affect the model training; for this reason, we filter the dataset as in [184], obtaining a subset of ∼82k objects. For each object, we render 16 images with azimuth evenly distributed from 0° to 360° and elevation view fixed to 30°.

To speed up fine-tuning, we first build the latent dataset by randomly sampling for every object a set of 4 images having azimuth offset by multiple of 90°; importantly, the latent variables are deterministically generated by using the mode of posterior distribution produced by the frozen encoder e_ϕ employed in the Stable Diffusion variational autoencoder.

We use the AdamW [185] optimizer with peak learning rate 5×10^{-5}, and cosine annealing with 100 warm-up steps. The total batch size is 112, with gradient accumulation over 12 batches, and image resolution is 256×256. We fine-tune our model for 15750 steps on 7 NVIDIA A100-80GB GPUs, which take about 5 days.

5.5.2 Evaluation

Baseline methods. We compare DiMViS to two state-of-the-art NVS 2D diffusion models, namely Zero123-XL and EscherNet [165], by using their publicly available pre-trained models. Zero123-XL inherits the architecture from Zero123 [159], but it is trained with the extended version of the Objaverse dataset, i.e., Objaverse-XL [186], which contains over 1 million 3D objects. Zero123-XL is a one-to-one model, as it accepts only one conditioning image to generate a single output representing the same object from a given different angle. Conversely, EscherNet is trained on the standard version of Objaverse-1.0 and is a many-to-many generative model, as it can accept more than one conditioning input image and generates multiple outputs from the target viewpoints. Specifically, we use the EscherNet variant employing 6 DoF camera poses. Indeed, both reference and target camera poses are required by EscherNet, and Zero123-XL; in contrast, DiMViS does not need such information,

as our implemented method generates a set of 4 images with azimuth evenly spaced between 0° and 360°; nevertheless, the set of potentially achievable viewpoints can be enlarged during training, such that DiMVIS could generate more views at inference.

Metrics. For the comparison, we compute three key metrics: the peak signal-to-noise ratio (PSNR), the structural similarity index measure (SSIM) [187] and the learned perceptual image patch similarity (LPIPS) [188]. PSNR is a widely used metric based on the pixel-wise difference between two images; SSIM measures the similarity of two images by comparing luminance, contrast, and structure. SSIM aims to reflect better the human visual perception, which is also the goal of LPIPS. The latter, though, computes the difference between the features obtained from a layer of a pre-trained image Convolutional Neural Network, namely the VGG [88] in our implementation.

Evaluation dataset. We evaluate DiMVIS and the baseline models on the Google Scanned Objects (GSO) [189] dataset. Specifically, we use the same subset of 30 objects used in [160, 165], by rendering 16 images for each object following the procedure described for the training set. Notably, for the rendering, we use the same lighting conditions used in [165], although they differ from the lighting settings of our training set, which are the same as in [160]. This poses an additional generalization challenge for DiMVIS.

For an extensive evaluation, we consider all possible subsets compatible with the fine-tuning dataset settings: to do so, we first build all four sets having four images with an azimuth offset by multiple of 90°, and, for each set, we finally consider all subsets with one, two and three conditioning reference views.

Results. Tab. 5.1 shows the considered metrics computed on the GSO dataset using the DDIM [190] sampler with 50 steps, as well as the values of the classifier-free guidance (CFG) [191] scale used for the generation. In particular, we compute all the metrics with CFG equal to 1 and 3, but we report only the best obtained results. Interestingly, we found that when having only one condition view, DiMVIS works better setting the CFG scale equal to 3, while with two and three conditioning images we obtain the best performance without using classifier-free guidance, i.e., setting the CFG scale equal to 1.

When considering only one reference view, DiMVIS works slightly better than Zero123-XL; furthermore, the latter is a one-to-one model, hence lacking flexibility, and the performance gap gets larger when DiMVIS is used with multiple reference

Table 5.1 Numerical evaluation of novel view synthesis on GSO

Method	Target views	Ref. views	CFG scale	PSNR (↑)	SSIM (↑)	LPIPS (↓)
Zero123-XL	3	1	3.0	16.13	0.772	0.181
EscherNet			3.0	14.85	0.755	0.209
DiMViS			3.0	<u>16.32</u>	<u>0.78</u>	<u>0.171</u>
EscherNet*	3×5		3.0	**17.26**	**0.8**	**0.165**
EscherNet	2	2	3.0	19.57	0.828	0.132
DiMViS			1.0	**21.07**	**0.854**	**0.113**
EscherNet*	2×7		3.0	<u>20.76</u>	<u>0.849</u>	<u>0.117</u>
EscherNet	1	3	3.0	20.20	0.821	0.131
DiMViS			1.0	**23.96**	**0.886**	**0.088**
EscherNet*	1×15		3.0	<u>22.35</u>	<u>0.869</u>	<u>0.104</u>

views. Overall, DiMViS also outperforms EscherNet. Interestingly, the latter does not show a clear improvement when increasing the number of reference views from 2 to 3. In this regard, EscherNet is empirically proven to work at its best with many target views, being them random or duplicates of the actual target one. For this reason, we evaluate EscherNet by repeating each target 5, 7, and 15 times when the actual number of targets is respectively 3, 2 and 1, and taking only the first output image. This workaround, referred to as "EscherNet*" in Tab. 5.1, leads to better performance at the expense of increased computational cost during inference, with EscherNet outperforming DiMViS only when a single view is available. Nevertheless, even in this case, DiMViS outperforms EscherNet when multiple views are available; we attribute the superior DiMViS performance to its capability of better aggregating the information provided by the reference images.

Fig. 5.2 shows the images generated using the considered methods. As it is possible to notice, the images produced by Zero123-XL are coherent with the (single) reference view, but are not consistent with each other, as they are generated independently. EscherNet overcomes this drawback by implementing specifically designed attention mechanisms, while DiMViS addresses this challenge by leveraging a joint diffusion model that exploits the self- and cross-attention mechanisms inherited from the pre-trained Stable Diffusion, without requiring any modification.

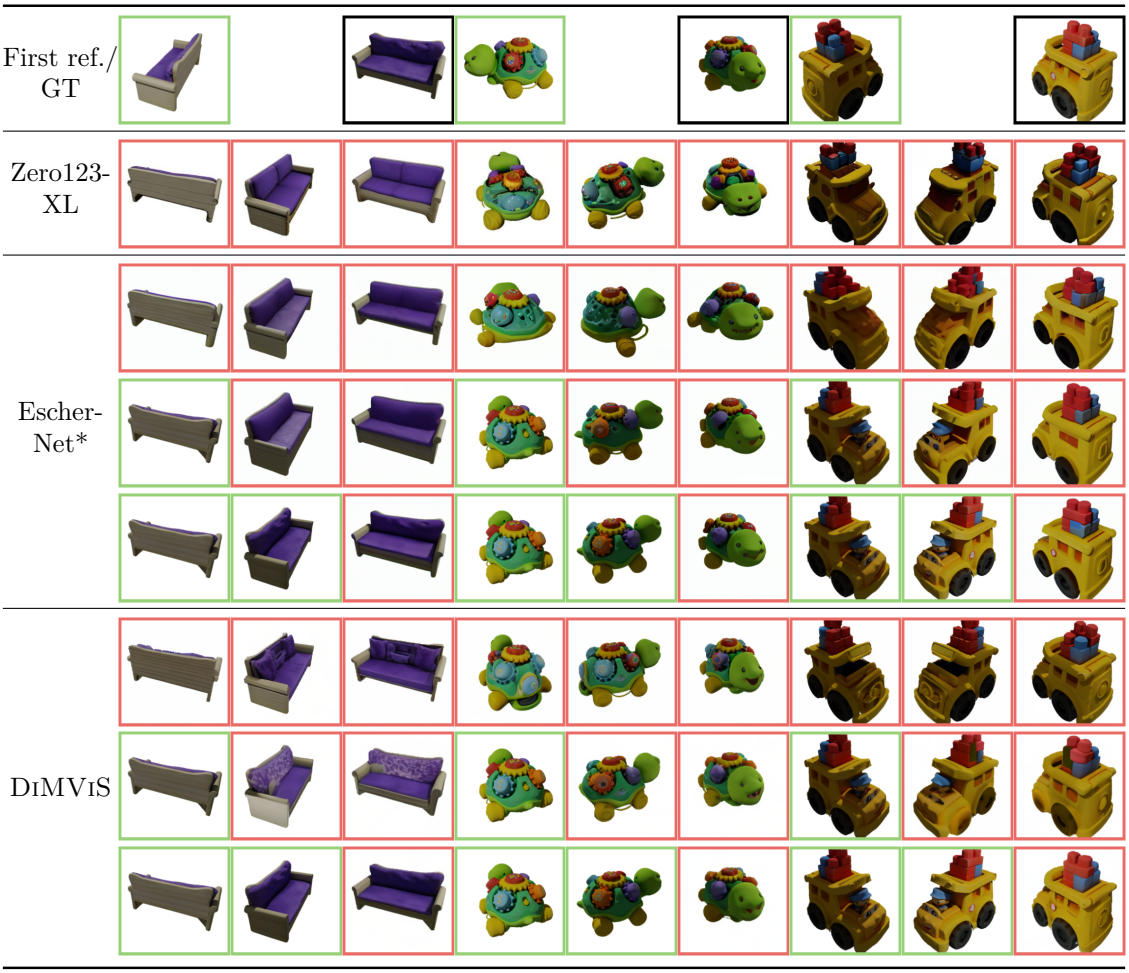


Fig. 5.2 **Visual evaluation of novel view synthesis visualization on GSO datasets.** Ground truth (GT), reference and generated images are respectively represented in black, green and red boxes. Note that, for each object, the first reference view is used for all generation processes, whereas we show only one ground truth image, as the other ones are already displayed as conditioning views. DiMVIS produces images that exhibit higher quality and diversity while ensuring reference-to-target and target-to-target consistency.

The images produced with DiMVIS generally exhibit higher visual quality and consistency; furthermore, DiMVIS seems to generate more diverse images throughout different runs (e.g., changing the set of conditioning images), while still maintaining reference-to-target and target-to-target consistency. Nevertheless, the shades produced by DiMVIS, even if plausible, do not always accurately reflect those present in the ground truth pictures: this is due to the different lighting conditions between our training set and the used test set, created following the same settings

employed in EscherNet paper. However, this issue is alleviated when more views are observed.

Consistency and diversity. Importantly, when increasing the number of reference views, more information is available and the generative process is more constrained: this means that the inherent randomness of the generation decreases and the produced novel views get closer to the ground-truth images, which leads to improved performance in terms of the analyzed metrics. However, the low metrics values obtained with fewer reference views do not necessarily indicate poor generation quality. Consider for example the sofa pictures in Fig. 5.2 obtained with DiMViS. When only one or two condition views are available, the diffusion model is less constrained and can be more imaginative, generating different images that are still plausible and realistic, and also consistent with each other and with the available views. However, the standard metrics used in NVS compute the similarity, either pixel-wise or perceptual, between the generated image and ground truth, thus penalizing the diversity in the produced images that are not close to the latter, but are perfectly coherent with the information provided by the reference view(s).

Nevertheless, diversity is one of the three main requirements in generative modeling [192]. Then, to take diversity into account, we argue that the currently used metrics should be integrated with new evaluation methods better suitable for novel view synthesis. One metric typically used to compare generative model performance is the Fréchet Inception Distance (FID) [193], which evaluates both quality and diversity. However, the FID score has recently been shown to present some limitations [194, 195], and hence we do not include it in our analysis. Interestingly, [176] introduce what they call a "3D consistency scoring", a novel metric that respects the above-mentioned criteria. However, it is not compliant with our sparse-view setting, as it is based on a NeRF model, whose training requires many views. Alternatively, one viable way would be to resort to human evaluation, which we defer to our future work.

5.6 Conclusion

In this chapter, we have presented DiMViS, a multi-view 2D diffusion model able to accommodate a varying number of reference and target images. DiMViS capitalizes on Stable Diffusion, inheriting its generalization capability, and on masked diffusion,

which permits learning the latent conditional distribution by means of a joint diffusion model. Our experimental evaluation has demonstrated the superiority of DIMVIS, compared to two state-of-the-art baselines.

Although DIMVIS is flexible in terms of the number of conditioning and generated views, it presents some limitations. First, the training set should be further extended, in order to consider views with different elevations, diverse lighting conditions and not limited to object-centric samples: this would enable the model to achieve an even superior generalization ability. Second, the developed model is trained on, and hence only produces, images with azimuth offset by multiples of 90°; however, including the related camera poses as an additional input for further flexibility is straightforward, and a similar technique implemented in [159] can be adopted. Finally, DIMVIS is designed to operate with a set of only four images, and scaling to a significantly larger number of reference and target views could pose implementation challenges. Therefore, additional research will be required to tackle this limit, which could hinder the application of the proposed method in real-world scenarios.

In the next chapter, we will employ a diffusion model for Novel View Synthesis with architecture similar to the one presented in this chapter, even though scaled down due to the employed dataset, which is smaller and less complex than the Objaverse dataset. In particular, the generative model is integrated into a classification system to augment the training set and, most notably, the inference pipeline, leading to higher test accuracy and increased robustness in edge-assisted classification systems.

Chapter 6

DiMViDA: Diffusion-based Multi-View Data Augmentation

We present DiMViDA, a Diffusion-based Multi-View Data Augmentation method built upon an innovative approach for Novel View Synthesis, which uses an extension of diffusion generative models that accepts any number of input views and that can generate any number of missing output views. In this work, our goal is to analyze the benefits of such a generative model in the context of object classification. Given a single input view, we compare the object classification performance of state-of-the-art models, namely ResNet18 and MobileNetV3, using the input view, versus its application to novel views synthesized by our generative model, using such synthetic views to augment the training set. Notably, differently from other works, we also adopt such a multi-view data augmentation method at inference. Our experimental findings illustrate that novel view synthesis can enhance object classification capabilities of CNNs; remarkably, we envision using our developed inference mechanism to increase resilience in edge-assisted classification systems.

Part of the work described in this chapter has already been published in **Di Giacomo, G.**, Franzese, G., Cerquitelli, T., Chiasserini, C. F., Michiardi, P. (2024). DiMViDA: Diffusion-based Multi-View Data Augmentation. In IEEE CAMAD 2024. IEEE.

6.1 Introduction

Generative models are used in a wide range of domains, such as image synthesis and natural language processing. Among the possible applications, data augmentation strategies based both on Generative Adversarial Networks (GANs) [196, 197] and diffusion models [198] have been adopted to produce synthetic data that are then used to enhance the capabilities of trained Deep Neural Networks (DNNs). For example, while standard data augmentation methods are based on color (e.g. change of brightness, contrast, saturation) and geometric (e.g., flipping, rotation) transformations, [199] leverages diffusion models to perform data augmentation by changing the semantics of the data, increasing their diversity.

Recently, there has been a growing interest in multi-modal [167, 200] and multi-view [159, 165, 201] diffusion models, which are able to generate multiple data representing the same concept using different modalities, e.g., image and text, or different views. In this work, given the effectiveness of data augmentation techniques based on generative models and the development of increasingly powerful multi-view diffusion models, we present DiMViDA, a Diffusion-based Multi-View Data Augmentation method that uses a Novel View Synthesis diffusion model to augment both the training and the evaluation pipeline in the context of a classification task.

Specifically, we consider a single-view dataset consisting of N samples, each representing a view of an object, and we augment it by generating for each sample three additional images of the same object from different viewpoints. To do so, we employ the latent diffusion model (LDM) introduced in [167], which applies multi-time masked diffusion to endow the model with conditional generation capability. Then, the resulting augmented set is used to train a classification model. At inference, we follow the same augmentation procedure also for the test set, producing a set of 4 views for each test sample. Next, given a set of multi-view images, we evaluate the classification model individually for each view and, finally, we compute the mean of the 4 outputs produced by the classification model to predict the samples' class. We corroborate the effectiveness of our method by performing experiments on a multi-view dataset using two state-of-the-art classification models, i.e., ResNet18 [202] and MobileNetV3 [203], both training the models from scratch and fine-tuning them starting from pre-trained weights. Our approach leads to improvements in classification accuracy up to 20% and improves robustness to failures in edge-assisted classification systems.

A similar approach was introduced in [204]; however, a GAN was used instead of a diffusion model and, remarkably, data augmentation is only carried out to extend the training set. In contrast, in our work, we also augment the inference pipeline, enabling us to achieve superior performance compared to only augmenting the training set.

The rest of the chapter is organized as follows. Sec. 6.2 discusses some related work, while Sec. 6.3 introduces the latent diffusion model utilized to generate novel synthetic views. Sec. 6.4 presents the designed methodology, while the experimental details and performance of DiMViDA are presented in Sec. 6.5. Finally, Sec. 6.6 draws our conclusions.

6.2 Related work

A comparison of the three main families of generative models, namely VAEs, GANs and diffusion models, is presented in [192], which identifies the three primary challenges in generative modeling: high-quality sample generation, sample diversity, and fast generation. This is referred to as the "generative learning trilemma". While GAN-based methods are characterized by a low diversity of generated data, and VAEs tend to produce samples of lower quality, score-based diffusion models are capable of generating both high-quality and diverse samples, though the generation process is slower.

Augmentation with synthetic data. Many works have implemented data augmentation techniques that rely on GANs for classification tasks, such as [196, 197]. [205] trains a classifier using only synthetic data produced with a GAN, resulting in improved performance compared to the original dataset. [204] proposes a method similar to ours, generating an augmented dataset by producing images of real data from novel viewpoints; however, data augmentation is performed by means of a GAN and, more importantly, only involves the training set. Our approach, instead, takes advantage of the multi-view synthetic data also during evaluation.

Recently, diffusion models have replaced GANs for data augmentation approaches based on synthetic images, thanks to their superior generation capability, both in terms of data quality and diversity. For instance, [198] demonstrates the benefits of using synthetic images generated by a state-of-the-art text-to-image diffusion

model in the context of image classification. In particular, the impact of synthetic data is analyzed in three scenarios, namely zero-shot learning, few-shot learning and large-scale model pre-training for transfer learning. [199] leverages diffusion models to perform data augmentation by changing the data semantics, increasing data diversity.

6.3 Latent diffusion model

In this section, we summarize the architecture of our Novel View Synthesis generative model, namely a latent diffusion model, which is composed of a deterministic autoencoder and a score-based diffusion model. Note that in this section – and in Sec. 6.4, as well – some details are similar to those presented in Chapter 5. However, we report them for the sake of clarity and to better point out the differences.

Autoencoder. In contrast to the work in Chapter 5, we have to train the autoencoder. We recall that the autoencoder consists of two parts: an encoder e_ϕ that produces a compact representation of the input, i.e., its latent variable, and the decoder d_ψ, which maps back such latent into the input space. During training, performed independently and prior to the diffusion model, the autoencoder learns to compress and recreate data, aiming to minimize the difference between the input and the final output. Thus, given a data distribution $p(x)$, we minimize the following objective function:

$$\mathcal{L} = \int p(x) l(x - d_\psi(e_\phi(x)))\, dx\,, \tag{6.1}$$

where l is the desired distance function.

Score-based diffusion model. After training the deterministic autoencoder, the encoder is used to produce latent representations of the data. Such latent variables are then employed to train the score-based diffusion model, which allows the model to learn the latent distribution, enabling the conditional generation capability at inference time. For further details, refer to Sec. 5.3. Notably, differently from the model presented in Chapter 5, we don't use the CLIP model and we keep from [167] the *multi-time vector* $\tau{=}[t_1, \ldots, t_V]$, which has two purposes, as it indicates both the diffusion time for the respective portions of latent variables and denotes the missing views. Formally, we define the multi-time vector as $\tau(M, t){=}t\left[\mathbb{1}(1{\in}M), \ldots, \mathbb{1}(V{\in}M)\right]$, where M is the sets of indices of the missing

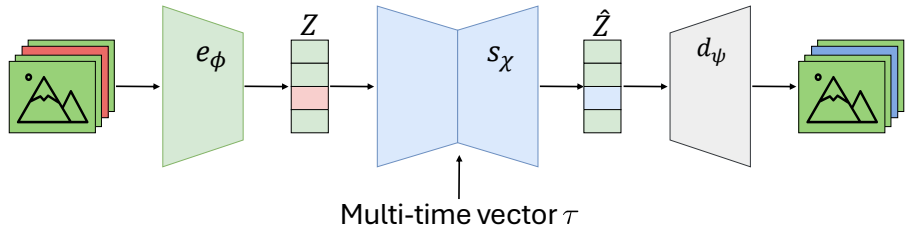


Fig. 6.1 Architecture of the employed latent diffusion model.

views, i.e., the views that should be generated. Notably, we employ a continuous-time diffusion formulation as in [167].

6.4 Methodology

In this section, we explain in detail our methodology, which relies on the latent diffusion model built on the method introduced in [167], to generate synthetic views that are used to improve the performance of a classification model.

6.4.1 Latent diffusion model

At inference time, the architecture of the latent diffusion model is depicted in Fig. 6.1. We consider a set of $V{=}4$ views in total and, for instance, the set of missing views with $M{=}\{3\}$. Also, we denote with C the set of indices of available conditioning views; thus, in this example, $C{=}\{1,2,4\}$. The deterministic encoder e_ϕ encodes each conditioning reference view X^c producing their latent representations $Z^c{=}e_\phi(X^c)$, while the missing view latent variable is represented using random noise, by sampling from the Normal distribution $\mathcal{N}(0,I)$. The obtained latent representations are concatenated in order to obtain the variable Z, which is fed, together with the multi-time vector τ, into the score-based diffusion model s_χ. Finally, the diffusion model produces the latent variable $\hat{Z}^m$ of the missing view X^m, which is given in input to the deterministic decoder d_ψ to generate the image $\hat{X}^m{=}d_\psi(\hat{Z}^m)$.

Training procedure. First, the encoder e_ϕ and the decoder d_ψ are jointly trained by optimizing the loss function in (6.1). Then, they are frozen, and we train the score-based diffusion model, which learns the conditional distribution of the missing latent variables $Z^M{=}\{Z^m\}_{m\in M}{\sim}p(z^M|z^C)$. For the training, we follow the

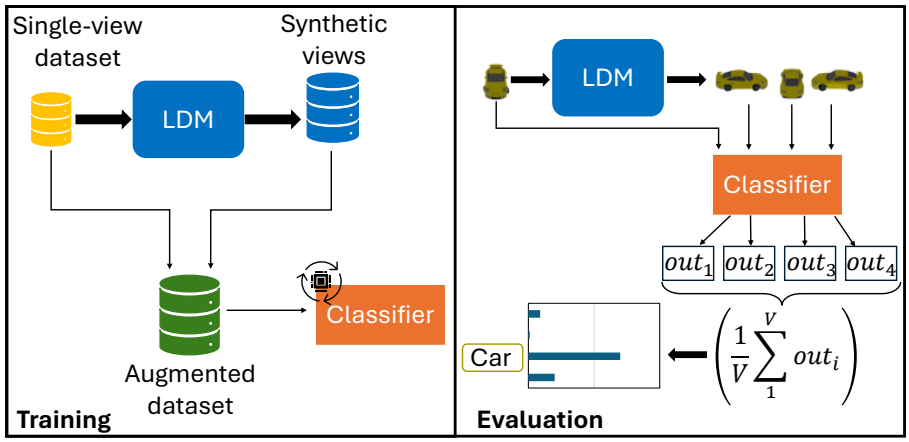


Fig. 6.2 Scheme of DiMViDA for training (left) and evaluation (right).

same procedure described in Sec. 5.4.1, except for the CLIP embeddings, which are not used in this model, and for the multi-time vector τ, which is built as explained in Sec. 6.3.

Formally, we train the score-based diffusion model s_χ by minimizing the following loss:

$$\mathcal{L} = \lambda(M,C) \cdot \left\| \mathcal{M}(M) \odot \left[\nabla \log\Big(q(R_t \mid z^C)\Big) - s_\chi(R_t, \tau(M,t)) \right] \right\|_2^2. \tag{6.2}$$

We remark that, as for (5.5), to obtain R_t we first sample from the distribution $q(r \mid Z, t)$ and then we aggregate the input Z using the mask $\mathcal{M}(M)$. Also in this case, the factor $\lambda(M,C)=1+\frac{|C|}{|M|}$ is introduced in order to take into account the diffusion of different portions of the latent variable, given by the randomization of M and C.

6.4.2 Classification pipeline

Once the latent diffusion model is trained, it is endowed with the capacity to generate missing views given the observed ones. Thus, we exploit this conditional generation ability to augment the classification pipeline.

The training procedure scheme of DiMViDA is presented in Fig. 6.2(left). Given a single-view dataset, that is, composed of N samples with one view per sample, we use the LDM to generate for each sample 3 other images from different viewpoints. Then, we train a classification model using such an augmented training set. As

shown in Fig. 6.2(right), during the evaluation step, we also augment the test set, following the same procedure used for the training set; thus, for each sample in the test data we produce a set of 4 views, including the real image and 3 synthetic ones. Next, for each set of multi-view images, we evaluate the model individually for every view and we compute the mean of the 4 classification model outputs, which is finally used to predict the class of the sample.

6.5 Experiments

In this section, we first describe the dataset used for the experiments and we evaluate the generation performance of the diffusion model. Finally, we assess the effectiveness of our augmented classification pipeline.

Datasets. We assess DiMViDA performance using the Neural 3D Mesh Renderer Dataset (NMR) [206]. NMR consists of objects of the 13 largest classes of ShapeNet [207] dataset, a collection of 3D objects, from which 64x64 2D images are rendered at 24 fixed views. For our experiments, we only use 4 views, i.e., front, back, right and left views.

Specifically, the dataset is split into three partitions:

- NMR_d, it includes for each sample all 4 views and is used to train both the autoencoder and the diffusion model;
- NMR_c, it includes only one random view per sample and is used to train the classifiers;
- NMR_e, it includes only one random view per sample and is used to evaluate the classifiers.

6.5.1 Training details of latent diffusion model

For the autoencoder, we use the same architecture used in [167] for the CUB [208] dataset and train the model using the Laplace distribution to estimate the likelihood. We used *TrivialAugmentWide* from the Torchvision library for data augmentation. We set the dimensionality of the latent space to 64; we perform 1000 training epochs, with learning rate 1e−4 and batch size equal to 64. Regarding the score-based

Table 6.1 Performance of latent diffusion model

	Available images		
Metric	1	2	3
FID (↓)	16.43	15.14	14.51
LPIPS (↓)	0.074	0.056	0.048
SSIM (↑)	0.832	0.884	0.912
PSNR (↑)	24.028	26.776	28.491
Coherence (%↑)	85.49	88.33	89.14

diffusion model, we borrow the architecture from [167] used for the CUB dataset. However, we extend the input dimension from 1024 to 1536. During training, we perform 1000 epochs, with the learning rate and batch size, respectively, equal to 1e−4 and 64.

6.5.2 Results

Conditional image generation performance. First, we evaluate the performance of the LDM by assessing the quality of the conditionally generated images by computing the Fréchet Inception Distance (FID) [193], Peak Signal-to-Noise Ratio (PSNR), the Structural Similarity Index Measure (SSIM) [187] and the Learned Perceptual Image Patch Similarity (LPIPS) [188]. The FID score evaluates both the quality and diversity of the generated data; the other three metrics, i.e., PSNR, SSIM and LPIPS, have been already introduced in Sec. 5.5.2. Importantly, for the LPIPS we employed the SqueezeNet [209] Convolutional Neural Network to extract the activations used to compute the similarity.

Furthermore, we verify that such generated data preserve the semantics of the conditioning images, i.e., the observed images: to do so, we measure the so-called coherence. Specifically, we use a pre-trained classifier Γ fine-tuned on our dataset to check that the generated images are classified coherently with the conditioning images. Formally, for N generated images, the coherence is computed as follows:

$$coherence(\hat{X}^m|X^C) = \frac{1}{N}\sum_{1}^{N} \mathbb{1}_{\{\Gamma(\hat{X}^m)=y_{X^C}\}}, \tag{6.3}$$

Table 6.2 Classifiers accuracy (%) – Training from scratch

Architecture	Model - training set	Test set	
		NMR_e	NMR_{ea}
ResNet18	Ω_s - NMR_c	83.89	87.21
	Ω_m - NMR_{ca}	85.18	**88.03**
MobileNetV3	Ω_s - NMR_c	71.01	79.66
	Ω_m - NMR_{ca}	77.95	**85.45**

Table 6.3 Classifiers accuracy (%) – Fine-tuning

Architecture	Model - training set	Test set	
		NMR_e	NMR_{ea}
ResNet18	Ω_s - NMR_c	89.98	89.89
	Ω_m - NMR_{ca}	90.83	**90.96**
MobileNetV3	Ω_s - NMR_c	87.14	88.15
	Ω_m - NMR_{ca}	88.08	**90.00**

where $\hat{X}^m$ is the image generated by the diffusion model conditionally with respect to set X^C, while y_{X^C} is the true label, i.e., the class of the conditioning set.

The computed metrics are reported in Tab. 6.1, which shows that the performance improves when more images are observed, highlighting that the model efficiently aggregates information from different views. In general, the reported metrics also indicate the capability of the latent diffusion model to generate high-quality data, while maintaining the semantics of the conditioning images when generating the missing views.

Classification performance. To evaluate the impact of synthetic images on the augmented classification pipeline, we first consider a classifier Ω_s, trained on the single-view NMR_c dataset, and a classifier Ω_m, trained on the augmented dataset NMR_{ca}, which includes both all NMR $images_c$ and the synthetic images obtained with the diffusion model conditioned on NMR_c data. During classifier training, we use the Adam optimizer with learning rate 1e−5 and batch size set to 128.

As for the evaluation protocol, we consider two approaches. The first is the standard procedure, where trained models are tested on the NMR_e single-view

Table 6.4 Accuracy (%) - Augmented vs. real views.

	Architecture	Model - training set	Test set	
			Real	NMR_{ea}
Training from scratch	ResNet18	Ω_s - NMR_c	88.97	87.21
		Ω_m - NMR_{ca}	89.73	88.03
	MobileNetV3	Ω_s - NMR_c	80.55	79.66
		Ω_m - NMR_{ca}	85.61	85.45
Fine-tuning	ResNet18	Ω_s - NMR_c	93.43	89.89
		Ω_m - NMR_{ca}	93.23	90.96
	MobileNetV3	Ω_s - NMR_c	91.79	88.15
		Ω_m - NMR_{ca}	92.56	90.00

evaluation data set. The second method uses the latent diffusion model to augment the NMR_e dataset, as done with NMR_c, obtaining the dataset NMR_{ea}.

We perform experiments with two state-of-the-art classification models, namely ResNet18 and MobileNetV3, training them from scratch. The results reported in Tab. 6.2 underline that the use of augmented multiview datasets both for training and evaluation leads to the best classification performance. Moreover, we also find that performing the augmentation only during the evaluation improves the performance more than augmenting solely the training set.

Finally, we also fine-tune the models starting from the pre-trained weights provided by the Torchvision library. The obtained results are shown in Tab. 6.3, which further demonstrates the benefits of our augmented classification pipeline, even if the gain is lower than the case of the training from scratch: very likely, the initialization of the model with the pre-trained weights equips the classification models with an already appropriate visual understanding ability, limiting further enhancements.

Results implications. Our results have corroborated the effectiveness of synthetic data for augmentation during training. Furthermore, unlike other works, we have shown how synthetic data can also be useful for inference. This opens interesting possibilities: specifically, we envision using the designed inference approach to make edge-assisted classification systems more robust to failures. For instance, we can consider a scenario with four cameras that capture the same scene from different viewpoints, offloading the classification task to an edge node. In case of camera or

transmission failures, the edge node may receive only one view; if the missing views are generated by using the diffusion model, higher classification accuracy can be achieved, compared to using only the single received view, as shown by the results present in Tab. 6.2 and Tab. 6.3. In this context, the generative model enhances the system's resilience, hence yielding increased reliability in case of failures.

Finally, we also show the performance of the system when there are no failures, hence with the edge node receiving all four real observed views. To this end, Tab. 6.4 compares the accuracy obtained with our multi-view inference scheme between the two scenarios without and with failures; in the latter situation, the edge node uses the single received image together with three synthetic views (NMR_{ea}): as expected, in this case, the accuracy is lower; in particular, the gap is small if models are trained from scratch, and slightly larger when fine-tuning them. However, by employing the synthetic images we could achieve significant benefits from the communication perspective. Indeed, our generative-based inference method can work with only one transmitted image, saving a significant amount of data compared to the scenario where all four views must be sent.

6.6 Conclusions

In this chapter, we have presented DiMViDA, a novel data augmentation technique based on a diffusion-based novel view synthesizer method. Our approach is designed for a classification task: specifically, we envision augmenting data both during training and inference, as the evaluation step takes into account the single real test image and the ones generated by the diffusion model, which represent the same object from different viewpoints. Importantly, the experiments performed have supported the effectiveness of DiMViDA, with gains in classification accuracy up to 20%. Finally, we also highlighted that the proposed inference method makes edge-assisted classification systems more resilient.

This chapter concludes the second part of this manuscript. In the following one, we will summarize the presented works and highlight our main findings. Finally, the closing section will discuss the direction for future research and the open challenges.

Chapter 7

Conclusions

In the first part of this thesis, we presented Cooperative Learning as a powerful paradigm suitable for many scenarios – especially those where data privacy is a major concern – and we analyzed some of the key challenges of this technique. Primarily, in Chapter 2 and Chapter 3, we addressed the energy minimization problem at training time in two different scenarios. In Chapter 2 the DNN layers are distributed across the learning nodes, which thus keep only a subset thereof; in Chapter 3, instead, the nodes sequentially train the whole DNN: notably, throughout the learning the model can be pruned to allow lower-capability nodes to participate in the training. Importantly, the many factors influencing the learning affect one another, thus the decisions about them must be made jointly; to this end, we have proposed two algorithmic frameworks that have been proven to achieve near-optimal solutions, leading to reduced energy consumption while meeting time and performance constraints.

In Chapter 4, we addressed another issue of CL, namely the need for an incentive mechanism to encourage learning nodes to collaborate during the model training. To cope with this, we proposed a game-theoretic algorithm that attains the Nash Equilibrium and converges to the fair Pareto optimal operating point by leveraging the concept of generosity.

In the second part of this manuscript, we focused on GenAI: generative models can generate high-quality synthetic data, which have risen as an effective alternative to real data in a variety of domains; for instance, synthetic data have been used to improve the performance of DNNs. For this reason, in Chapter 5 we designed a diffusion model for Novel View Synthesis: specifically, it operates in the context

of multi-view images and is capable of generating multiple novel views of an object conditioned on a variable number of observed ones. To achieve this, our model leverages the masked diffusion approach, built upon a pre-trained model, which allows it to outperform existing state-of-the-art models by up to 15% in perceptual similarity. Finally, in Chapter 6 we also demonstrated that incorporating such a Novel View Synthesis model into an image classification pipeline can significantly enhance the final accuracy, by up to 20%. To this end, we augmented the training set with synthetic views and designed a method that combines a real observed view with synthetic ones at inference time. Remarkably, the latter technique could be applied for edge-assisted classification tasks: the edge node, to which the inference task is offloaded, can use the generative model to recover potentially missing data, hence increasing the resilience of the overall classification system.

7.1 Future work and open challenges

Generative Artificial Intelligence is already revolutionizing various fields, raising widespread interest from academia and industry. In our work, we empirically proved the utility of synthetic data; thus, we will further explore the vast potential of generative models. In this section, we discuss our future research direction, and we will highlight some of the open challenges that research should address.

Metrics for generative models. In Chapter 5 we pointed out the limitations of the metrics used to evaluate the diversity of the generated views and their consistency with the available conditioning views. Assessing the quality of images produced by generative models is difficult even considering the standard (single-view) case. For instance, [210] finds that the current metrics do not align well with human judgment, proposing a better alternative. Therefore, future research should focus on improving evaluation metrics to reflect the performance of generative models more accurately. As for our Novel View Synthesis model, we will study new evaluation methods tailored specifically for the multi-view scenario.

Video Diffusion Models. Video Diffusion Models are a novel trend in GenAI, as they extend the principle of models operating solely in the image space by also handling the time dimension. Video Diffusion Models are able to capture the time dependency among consecutive images, which endows them with the capability to produce a sequence of temporally coherent frames. Therefore, extending our

multi-view diffusion model to support video generation video could further increase its flexibility and, hence, its applicability in real use cases.

GenAI and Cooperative Learning. Synthetic data produced by generative models could be useful also in CL scenarios. For instance, [211] presents a data augmentation framework for image classification in a distributed learning setting. Specifically, Federated Learning is coupled with a pre-trained generative model running at the parameter server, which produces synthetic data and sends them to the devices in order to mitigate the data heterogeneity issue, i.e., local data are non-IID across clients. The approach takes into account the constraints on the local resources and outperforms the standard Federated Learning algorithm, achieving lower devices' energy consumption, shorter training time and improved accuracy.

On the other hand, also Cooperative Learning schemes could be useful for GenAI, as they can be applied to train, or better fine-tune, generative models. This was envisioned, for example, in [212], which demonstrates the advantages of federated fine-tuning in terms of communication overhead and time compared to the centralized strategy. Additionally, Cooperative Learning may be the only feasible approach in those domains, such as healthcare, where strict privacy constraints prohibit sharing data with third parties.

Despite the advancements in each field, the potential of combining Cooperative Learning and Generative AI remains largely unexplored; thus, further research is essential to deeply study how these two techniques could be merged, investigating the mutual benefits and the opportunities their synergy could unlock.

Sustainability. Training generative models from scratch may incur significant costs due to the large number of parameters and training samples required to achieve good performance. Furthermore, due to the scale of such models, inference can be costly as well. Consequently, sustainability is one of the major concerns for GenAI and a detailed analysis of costs and benefits is essential when deploying these models into real-world applications.

One typical approach to alleviate the training costs is employing large pre-trained models, which deliver excellent performance and generalization capabilities, requiring minimal or no fine-tuning to be adapted to the target task. To further reduce time, computational and energy consumption during this stage, many efficient yet effective fine-tuning techniques have been introduced, such as Low-Rank Adaptation (LoRA). Also, it could be interesting to develop and apply methods similar to the one

proposed in Chapter 3 to cooperatively fine-tune generative models. Furthermore, other well-known model compression techniques, e.g., pruning, quantization and knowledge distillation, can be utilized to obtain smaller-scale models that require reduced resources for inference, increasing efficiency.

Despite the progress in this area, GenAI still has a significant environmental footprint, necessitating urgent solutions to enable a sustainable and responsible deployment.

Explainability. An increasingly important research topic in ML is explainability, which aims to understand how models work and make decisions, to increase their transparency to the users. As for GenAI, exploring the behavior of the models and identifying the reasons leading to a certain output is crucial for providing guarantees about the quality and reliability of the generated data, especially when integrating generative models in practical implementations.

References

[1] The Nobel Prize in Physics 2024. Nobel Prize Outreach AB 2024. Tue. 29 Oct 2024. https://www.nobelprize.org/prizes/physics/2024/summary/. Accessed: 2024-10-29.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012.

[3] ImageNet Large Scale Visual Recognition Challenge (ILSVRC) . https://https://www.image-net.org/challenges/LSVRC/. Accessed: 2024-11-25.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.

[5] Qinbin Li, Zeyi Wen, Zhaomin Wu, et al. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[6] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE transactions on information forensics and security*, 15:3454–3469, 2020.

[7] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[8] Francesco Malandrino, Carlos Barroso Fernandez, Carlos J Bernardos Cano, Carla Fabiana Chiasserini, Antonio De La Oliva, and Mahyar Onsori. Data-driven and privacy-preserving cooperation in decentralized learning. In *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, pages 1–8. IEEE, 2024.

[9] Michael Terrell. New nuclear clean energy agreement with Kairos Power. https://bit.ly/3UHK9Iw. Accessed: 2024-08-11.

[10] Shekoofeh Azizi, Simon Kornblith, Chitwan Saharia, Mohammad Norouzi, and David J. Fleet. Synthetic data from diffusion models improves imagenet classification. *Transactions on Machine Learning Research*, 2023.

[11] Yaad Oren and Francesco Di Cerbo. How synthetic data can avoid company risk with AI. https://www.sap.com/blogs/how-synthetic-data-can-avoid-company-risk-with-ai. Accessed: 2024-08-11.

[12] Ella Peltonen, Mehdi Bennis, Michele Capobianco, Merouane Debbah, Aaron Ding, Felipe Gil-Castiñeira, Marko Jurmu, Teemu Karvonen, Markus Kelanti, Adrian Kliks, et al. 6g white paper on edge intelligence. *arXiv preprint arXiv:2004.14850*, 2020.

[13] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165, 2019.

[14] Luciano Baresi, Danilo Filgueira Mendonça, Martin Garriga, Sam Guinea, and Giovanni Quattrocchi. A unified model for the mobile-edge-cloud continuum. *ACM Transactions on Internet Technology*, 2019.

[15] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

[16] Armin Kappeler, Robin D Morris, Amar Ramesh Kamat, Nikhil Rasiwasia, and Gaurav Aggarval. Combining deep learning and unsupervised clustering to improve scene recognition performance. In *IEEE PIMRC MMSP Workshop*, 2015.

[17] Chengxu Zhuang, Alex Lin Zhai, and Daniel Yamins. Local aggregation for unsupervised learning of visual embeddings. In *IEEE/CVF ICCV*, 2019.

[18] Massimiliano Mancini, Lorenzo Porzi, Samuel Rota Bulo, Barbara Caputo, and Elisa Ricci. Inferring latent domains for unsupervised deep domain adaptation. *IEEE Transactions on pattern analysis and machine intelligence*, 2019.

[19] Mohammad Reza Loghmani, Luca Robbiano, Mirco Planamente, Kiru Park, Barbara Caputo, and Markus Vincze. Unsupervised domain adaptation through inter-modal rotation for rgb-d object recognition. *IEEE Robotics and Automation Letters*, 2020.

[20] Mirco Planamente, Chiara Plizzari, Marco Cannici, Marco Ciccone, Francesco Strada, Andrea Bottino, Matteo Matteucci, and Barbara Caputo. Da4event: towards bridging the sim-to-real gap for event cameras using domain adaptation. *arXiv preprint arXiv:2103.12768*, 2021.

[21] Abdullatif Albaseer, Bekir Sait Ciftler, Mohamed Abdallah, and Ala Al-Fuqaha. Exploiting unlabeled data in smart cities using federated edge learning. In *IEEE IWCMC*, 2020.

[22] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *ACM/IEEE IPSN*, 2016.

[23] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems*, 2020.

[24] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[25] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 2018.

[26] Yansong Gao, Minki Kim, Sharif Abuadbba, Yeonjae Kim, Chandra Thapa, Kyuyeon Kim, Seyit A Camtepe, Hyoungshick Kim, and Surya Nepal. End-to-end evaluation of federated learning and split learning for internet of things. *arXiv preprint arXiv:2003.13376*, 2020.

[27] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 2020.

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

[29] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 2019.

[30] Mattia Merluzzi, Paolo Di Lorenzo, Sergio Barbarossa, and Valerio Frascolla. Dynamic computation offloading in multi-access edge computing via ultra-reliable and low-latency communications. *IEEE Transactions on Signal and Information Processing over Networks*, 2020.

[31] Jakub Konečný, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[32] Jiawen Kang, Zehui Xiong, Dusit Niyato, Yuze Zou, Yang Zhang, and Mohsen Guizani. Reliable federated learning for mobile networks. *IEEE Wireless Communications*, 2020.

[33] Yufeng Zhan, Peng Li, Zhihao Qu, Deze Zeng, and Song Guo. A learning-based incentive mechanism for federated learning. *IEEE Internet of Things Journal*, 2020.

[34] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Blockchained on-device federated learning. *IEEE Communications Letters*, 2019.

[35] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2020.

[36] Hung T Nguyen, Vikash Sehwag, Seyyedali Hosseinalipour, Christopher G Brinton, Mung Chiang, and H Vincent Poor. Fast-convergent federated learning. *IEEE Journal on Selected Areas in Communications*, 2020.

[37] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *International Conference on Learning Representations*, 2019.

[38] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 2019.

[39] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 2019.

[40] Yunlong Mao, Shanhe Yi, Qun Li, Jinghao Feng, Fengyuan Xu, and Sheng Zhong. A privacy-preserving deep learning approach for face recognition with edge computing. In *USENIX HotEdge*, 2018.

[41] Liekang Zeng, En Li, Zhi Zhou, and Xu Chen. Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things. *IEEE Network*, 2019.

[42] Thaha Mohammed, Carlee Joe-Wong, Rohit Babbar, and Mario Di Francesco. Distributed inference acceleration with adaptive dnn partitioning and offloading. In *IEEE INFOCOM*, 2020.

[43] Emna Baccour, Aiman Erbad, Amr Mohamed, Mounir Hamdi, and Mohsen Guizani. Distprivacy: Privacy-aware distributed deep neural networks in iot surveillance systems. In *IEEE GLOBECOM*, 2020.

[44] Tareq Si Salem, Gabriele Castellano, Giovanni Neglia, Fabio Pianese, and Andrea Araldo. Towards inference delivery networks: Distributing machine learning with optimality guarantees. In *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, pages 1–8, 2021.

[45] Mahmoud Assran, Joshua Romoff, Nicolas Ballas, Joelle Pineau, and Michael Rabbat. Gossip-based actor-learner architectures for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 32:13299–13309, 2019.

[46] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. Edge Intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 2020.

[47] Marc Barcelo, Alejandro Correa, Jaime Llorca, Antonia M. Tulino, Jose Lopez Vicario, and Antoni Morell. Iot-cloud service optimization in next generation smart environments. *IEEE Journal on Selected Areas in Communications*, 34(12):4077–4090, 2016.

[48] Hao Feng, Jaime Llorca, Antonia M. Tulino, Danny Raz, and Andreas F. Molisch. Approximation algorithms for the nfv service distribution problem. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.

[49] Matthias Rost and Stefan Schmid. Virtual network embedding approximations: Leveraging randomized rounding. *IEEE/ACM Transactions on Networking*, 27(5):2071–2084, 2019.

[50] Marcelo Michael, Jaime Llorca, and Antonia Tulino. Approximation algorithms for the optimal distribution of real-time stream-processing services. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–7, 2019.

[51] Konstantinos Poularakis, Jaime Llorca, Antonia M Tulino, and Leandros Tassiulas. Approximation algorithms for data-intensive service chain embedding. In *Proceedings of the Twenty-First International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pages 131–140, 2020.

[52] Francesco Malandrino, Carla Fabiana Chiasserini, and Giuseppe Di Giacomo. Energy-efficient training of distributed dnns in the mobile-edge-cloud continuum. In *IEEE/IFIP WONS*, 2022.

[53] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 2018.

[54] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, 2014.

[55] Jorge Martín-Peréz, Francesco Malandrino, Carla-Fabiana Chiasserini, and Carlos J Bernardos. Okpi: All-kpi network slicing through efficient resource allocation. In *IEEE INFOCOM*, 2020.

[56] Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 2019.

[57] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 2020.

[58] Xinxin Mei, Xiaowen Chu, Hai Liu, Yiu-Wing Leung, and Zongpeng Li. Energy efficient real-time task scheduling on cpu-gpu hybrid clusters. In *IEEE INFOCOM*. IEEE, 2017.

[59] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[60] Gek Hong Sim, Sabrina Klos, Arash Asadi, Anja Klein, and Matthias Hollick. An online context-aware machine learning algorithm for 5g mmwave vehicular communications. *IEEE/ACM Transactions on Networking*, 2018.

[61] Cristina Rottondi, Luca Barletta, Alessandro Giusti, and Massimo Tornatore. Machine-learning method for quality of transmission prediction of unestablished lightpaths. *Journal of Optical Communications and Networking*, 2018.

[62] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[63] Hiromitsu Takahashi and Akira Matsuyama. An approximate solution for the steiner problem in graphs. *Math. Japonica*, pages 573–577, 1980.

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

[65] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[66] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[67] Dirk G Cattrysse and Luk N Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European journal of operational research*, 1992.

[68] Alaa Awad Abdellatif, Carla Fabiana Chiasserini, and Francesco Malandrino. Active learning-based classification in automated connected vehicles. In *IEEE INFOCOM PERSIST-IoT Workshop*, 2020.

[69] Claudia Perlich, Foster Provost, and Jeffrey S Simonoff. Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research*, 2003.

[70] Hao Yi Ong, Kevin Chavez, and Augustus Hong. Distributed deep q-learning. *CoRR*, 2015.

[71] NVIDIA A100 datasheet. https://bit.ly/3O6jWxj. Accessed: 2021-07-30.

[72] NVIDIA RTX A4000 datasheet. https://bit.ly/3ceeUS8. Accessed: 2021-07-30.

[73] Apple unveils all-new iPad Air with A14 Bionic. https://www.apple.com/newsroom. Accessed: 2021-07-30.

[74] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. srsLTE: An open-source platform for lte evolution and experimentation. In *ACM WiNTECH MobiCom Workshop*, 2016.

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

[76] Davide Callegaro and Marco Levorato. Optimal edge computing for infrastructure-assisted uav systems. *IEEE Transactions on Vehicular Technology*, 2021.

[77] Peyman Tehrani, Francesco Restuccia, and Marco Levorato. Federated deep reinforcement learning for the distributed control of nextg wireless networks. In *IEEE DySPAN*, 2021.

[78] Dong-Jun Han, Do-Yeon Kim, Minseok Choi, Christopher G Brinton, and Jaekyun Moon. SplitGP: Achieving Both Generalization and Personalization in Federated Learning. In *IEEE INFOCOM*, 2023.

[79] Enrico Russo, Maurizio Palesi, Salvatore Monteleone, Davide Patti, Andrea Mineo, Giuseppe Ascia, and Vincenzo Catania. Dnn model compression for iot domain-specific hardware accelerators. *IEEE Internet of Things Journal*, 9(9):6650–6662, 2022.

[80] Hanxi Guo, Qing Yang, Hao Wang, Yang Hua, Tao Song, Ruhui Ma, and Haibing Guan. Spacedml: Enabling distributed machine learning in space information networks. *IEEE Network*, 2021.

[81] Davide Callegaro, Francesco Restuccia, and Marco Levorato. Smartdet: Context-aware dynamic control of edge task offloading for mobile object detection. In *IEEE WoWMoM*, 2022.

[82] Anran Li, Lan Zhang, Juntao Tan, Yaxuan Qin, Junhao Wang, and Xiang-Yang Li. Sample-level data selection for federated learning. In *IEEE INFOCOM*, 2021.

[83] Tiffany Tuor, Shiqiang Wang, Bong Jun Ko, Changchang Liu, and Kin K Leung. Overcoming noisy and irrelevant data in federated learning. In *IEEE ICPR*, 2021.

[84] Shipeng Fu, Zhen Li, Kai Liu, Sadia Din, Muhammad Imran, and Xiaomin Yang. Model compression for iot applications in industry 4.0 via multiscale knowledge transfer. *IEEE Transactions on Industrial Informatics*, 16(9):6013–6022, 2020.

[85] Song Tang, Yuji Shi, Zhiyuan Ma, Jian Li, Jianzhi Lyu, Qingdu Li, and Jianwei Zhang. Model adaptation through hypothesis transfer with gradual knowledge distillation. In *IEEE/RSJ IROS*, 2021.

[86] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 2016.

[87] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 2021.

[88] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[89] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[90] Andrea Bragagnolo and Carlo Alberto Barbano. Simplify: A python library for optimizing pruned neural networks. *SoftwareX*, 17:100907, 2022.

[91] Edouard Yvinec, Arnaud Dapogny, Matthieu Cord, and Kevin Bailly. RED++: Data-Free Pruning of Deep Neural Networks via Input Splitting and Output Merging. *arXiv:2110.01397*, 2021.

[92] Yifan Gong, Zheng Zhan, Zhengang Li, Wei Niu, Xiaolong Ma, Wenhao Wang, Bin Ren, Caiwen Ding, Xue Lin, Xiaolin Xu, et al. A privacy-preserving-oriented dnn pruning and mobile acceleration framework. In *ACM GLSVLSI*, 2020.

[93] Mario Osta, Mohamad Alameh, Hamoud Younes, Ali Ibrahim, and Maurizio Valle. Energy efficient implementation of machine learning algorithms on hardware platforms. In *IEEE International Conference on Electronics, Circuits and Systems (ICECS*, pages 21–24, 2019.

[94] Chit Wutyee Zaw, Shashi Raj Pandey, Kitae Kim, and Choong Seon Hong. Energy-aware resource management for federated learning in multi-access edge computing systems. *IEEE Access*, 9:34938–34950, 2021.

[95] Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling. *Wiley GAMM-Mitteilungen*, 2021.

[96] Tahmid Abtahi, Amey Kulkarni, and Tinoosh Mohsenin. Accelerating convolutional neural network with fft on tiny cores. In *IEEE International Symposium on Circuits and Systems (ISCAS*, pages 1–4, 2017.

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

[98] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120*, 2013.

[99] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *Advances in neural information processing systems*, 2019.

[100] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[101] Florent Altché and Arnaud de La Fortelle. An LSTM network for highway trajectory prediction. In *IEEE International Conference on Intelligent Transportation Systems (ITSC*, pages 353–359, 2017.

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

[103] Broadcom VideoCore VI technical details. https://bit.ly/3z5bytK. Accessed: 2021-07-30.

[104] Filipe Rodrigues and Francisco C Pereira. Beyond expectation: Deep joint mean and quantile regression for spatiotemporal problems. *IEEE Transactions on Neural Networks and Learning Systems*, 31(12):5377–5389, 2020.

[105] Zijian Gao, Kele Xu, Bo Ding, Huaimin Wang, Yiying Li, and Hongda Jia. Knowru: Knowledge reusing via knowledge distillation in multi-agent reinforcement learning. *arXiv:2103.14891*, 2021.

[106] Tiantian Zhang, Xueqian Wang, Bin Liang, and Bo Yuan. Catastrophic interference in reinforcement learning: A solution based on context division and knowledge distillation. *arXiv:2109.00525*, 2021.

[107] Luca Saglietti and Lenka Zdeborová. Solvable model for inheriting the regularization through knowledge distillation. *arXiv:2012.00194*, 2020.

[108] Haiyan Yin and Sinno Jialin Pan. Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *Thirty-First AAAI conference on artificial intelligence*, 2017.

[109] Zhang-Wei Hong, Prabhat Nagarajan, and Guilherme Maeda. Collaborative inter-agent knowledge distillation for reinforcement learning. 2019.

[110] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2018.

[111] Chong Min John Tan and Mehul Motani. Dropnet: Reducing neural network complexity via iterative pruning. In *ICML*, 2020.

[112] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *ECCV*, 2018.

[113] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning. In *ACM ASPLOS*, 2020.

[114] Jinhyuk Park and Albert No. Prune your model before distill it. *arXiv:2109.14960*, 2021.

[115] Nima Aghli and Eraldo Ribeiro. Combining weight pruning and knowledge distillation for cnn compression. In *IEEE/CVF CVPR Workshops CVPRW*, 2021.

[116] Tim Menzies, Seung-Hwan Lim, Hui Guan, Xipeng Shen, and Lin Ning. Generalization of teacher-student network and cnn pruning. Technical report, North Carolina State University. Dept. of Computer Science, 2018.

[117] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *ECCV*, 2018.

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

[119] Yoshitomo Matsubara, Davide Callegaro, Sabur Baidya, Marco Levorato, and Sameer Singh. Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems. *IEEE Access*, 2020.

[120] Yuhao Zhou, Qing Ye, and Jian Cheng Lv. Communication-Efficient Federated Learning with Compensated Overlap-FedAvg. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[121] Ayush Chopra, Surya Kant Sahu, Abhishek Singh, Abhinav Java, Praneeth Vepakomma, Vivek Sharma, and Ramesh Raskar. Adasplit: Adaptive trade-offs for resource-constrained distributed deep learning. *arXiv:2112.01637*, 2021.

[122] Tanmoy Sen and Haiying Shen. A data and model parallelism-based distributed deep learning system in a network of edge devices. In *ICDCS*, 2022.

[123] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *arXiv:2103.04505*, 2021.

[124] Othmane Marfoq, Giovanni Neglia, Richard Vidal, and Laetitia Kameni. Personalized federated learning through local memorization. In *ICML*, 2022.

[125] Ahmed M Abdelmoniem, Atal Narayan Sahu, Marco Canini, and Suhaib A Fahmy. Resource-Efficient Federated Learning. *arXiv:2111.01108*, 2021.

[126] Francesco Malandrino and Carla Fabiana Chiasserini. Federated learning at the network edge: When not all nodes are created equal. *arXiv preprint arXiv:2101.01995*, 2021.

[127] Hongda Wu and Ping Wang. Fast-convergent federated learning with adaptive weighting. *IEEE Transactions on Cognitive Communications and Networking*, 2021.

[128] Ahmed Imteaj and M Hadi Amini. Fedar: Activity and resource-aware federated learning model for distributed mobile robots. In *IEEE ICMLA*, 2020.

[129] Fan Lai, Yinwei Dai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. FedScale: Benchmarking model and system performance of federated learning. In *ACM SIGOPS ResilientFL Workshop*, 2021.

[130] Lei Yang, Fulin Wen, Jiannong Cao, and Zhenyu Wang. Edgetb: A hybrid testbed for distributed machine learning at the edge with high fidelity. *IEEE Transactions on Parallel and Distributed Systems*, 33(10):2540–2553, 2022.

[131] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[132] Francesco Malandrino, Carla Fabiana Chiasserini, and Giuseppe Di Giacomo. Efficient distributed dnns in the mobile-edge-cloud continuum. *IEEE/ACM Transactions on Networking*, 2023.

[133] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[134] Francesco Paissan, Alberto Ancilotto, Alessio Brutti, and Elisabetta Farella. Scalable neural architectures for end-to-end environmental sound classification. In *IEEE ICASSP*, 2022.

[135] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[136] Nikita Zeulin, Olga Galinina, Nageen Himayat, Sergey Andreev, and Robert W Heath Jr. Dynamic Network-Assisted D2D-Aided Coded Distributed Learning. *arXiv:2111.14789*, 2021.

[137] Zixiang Chen, Yuan Cao, Difan Zou, and Quanquan Gu. How much over-parameterization is sufficient to learn deep relu networks? In *ICLR*, 2021.

[138] Mary Phuong and Christoph Lampert. Towards understanding knowledge distillation. In *PMLR International Conference on Machine Learning*, 2019.

[139] Arman Rahbar, Ashkan Panahi, Chiranjib Bhattacharyya, Devdatt Dubhashi, and Morteza Haghir Chehreghani. On the unreasonable effectiveness of knowledge distillation: Analysis in the kernel regime. *arXiv:2003.13438*, 2020.

[140] Francesco Malandrino, Giuseppe Di Giacomo, Armin Karamzade, Marco Levorato, and Carla Fabiaina Chiasserini. Matching DNN compression and cooperative training with resources and data availability. In *IEEE INFOCOM*, 2023.

[141] Xiangru Lian, Ce Zhang, Huan Zhang, et al. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. *Advances in Neural Information Processing Systems*, 2017.

[142] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, et al. A unified theory of decentralized SGD with changing topology and local updates. In *PMLR International Conference on Machine Learning*, 2020.

[143] Jiawen Kang, Zehui Xiong, Dusit Niyato, et al. Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory. *IEEE Internet of Things Journal*, 2019.

[144] Martin A Nowak and Karl Sigmund. Tit for tat in heterogeneous populations. *Nature*, 1992.

[145] Vikram Srinivasan, Pavan Nuggehalli, C.F. Chiasserini, et al. An analytical approach to the study of cooperation in wireless ad hoc networks. *IEEE Transactions on Wireless Communications*, 2005.

[146] Francesco Malandrino, Carla Fabiana Chiasserini, Nuria Molner, et al. Network support for high-performance distributed machine learning. *IEEE/ACM Transactions on Networking*, 2023.

[147] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, et al. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 2020.

[148] Xuezhen Tu, Kun Zhu, Nguyen Cong Luong, et al. Incentive mechanisms for federated learning: From economic and game theoretic perspective. *IEEE Transactions on Cognitive Communications and Networking*, 2022.

[149] Abdullah B Akbay and Junshan Zhang. Distributed learning with strategic users: A repeated game approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[150] Latif U Khan, Shashi Raj Pandey, Nguyen H Tran, et al. Federated learning for edge networks: Resource optimization and incentive mechanism. *IEEE Communications Magazine*, 2020.

[151] Ming Tang and Vincent WS Wong. An incentive mechanism for cross-silo federated learning: A public goods perspective. In *IEEE INFOCOM*, 2021.

[152] Kate Donahue and Jon Kleinberg. Optimality and stability in federated learning: A game-theoretic approach. *Advances in Neural Information Processing Systems*, 2021.

[153] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[154] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[155] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[156] Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. *Advances in neural information processing systems*, 34:1415–1428, 2021.

[157] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[158] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in Neural Information Processing Systems*, 35:25278–25294, 2022.

[159] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9298–9309, 2023.

[160] Yuan Liu, Cheng Lin, Zijiao Zeng, Xiaoxiao Long, Lingjie Liu, Taku Komura, and Wenping Wang. Syncdreamer: Generating multiview-consistent images from a single-view image. In *The Twelfth International Conference on Learning Representations*, 2024.

[161] Haohan Weng, Tianyu Yang, Jianan Wang, Yu Li, Tong Zhang, CL Chen, and Lei Zhang. Consistent123: Improve consistency for one image to 3d object synthesis. *arXiv preprint arXiv:2310.08092*, 2023.

[162] Shijie Li, Farhad G Zanjani, Haitam Ben Yahia, Yuki M Asano, Juergen Gall, and Amirhossein Habibian. Valid: Variable-length input diffusion for novel view synthesis. *arXiv preprint arXiv:2312.08892*, 2023.

[163] Lukas Höllein, Aljaž Božič, Norman Müller, David Novotny, Hung-Yu Tseng, Christian Richardt, Michael Zollhöfer, and Matthias Nießner. Viewdiff: 3d-consistent image generation with text-to-image models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[164] Shitao Tang, Jiacheng Chen, Dilin Wang, Chengzhou Tang, Fuyang Zhang, Yuchen Fan, Vikas Chandra, Yasutaka Furukawa, and Rakesh Ranjan. Mvdiffusion++: A dense high-resolution multi-view diffusion model for single or sparse-view 3d object reconstruction. *arXiv preprint arXiv:2402.12712*, 2024.

[165] Xin Kong, Shikun Liu, Xiaoyang Lyu, Marwan Taher, Xiaojuan Qi, and Andrew J Davison. Eschernet: A generative model for scalable view synthesis. *arXiv preprint arXiv:2402.03908*, 2024.

[166] Justin Pinkney. Stable diffusion image variations - a hugging face space by lambdalabs.

[167] Mustapha Bounoua, Giulio Franzese, and Pietro Michiardi. Multi-modal latent diffusion. *Entropy*, 26(4):320, 2024.

[168] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[169] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5741–5751, 2021.

[170] Hanwen Jiang, Zhenyu Jiang, Kristen Grauman, and Yuke Zhu. Few-view object reconstruction with unknown categories and camera poses. *International Conference on 3D Vision (3DV)*, 2024.

[171] Prune Truong, Marie-Julie Rakotosaona, Fabian Manhardt, and Federico Tombari. Sparf: Neural radiance fields from sparse and noisy poses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4190–4200, 2023.

[172] Hanwen Jiang, Zhenyu Jiang, Yue Zhao, and Qixing Huang. LEAP: Liberate sparse-view 3d modeling from camera poses. In *The Twelfth International Conference on Learning Representations*, 2024.

[173] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022.

[174] Zhizhuo Zhou and Shubham Tulsiani. Sparsefusion: Distilling view-conditioned diffusion for 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12588–12597, 2023.

[175] Rundi Wu, Ben Mildenhall, Philipp Henzler, Keunhong Park, Ruiqi Gao, Daniel Watson, Pratul P Srinivasan, Dor Verbin, Jonathan T Barron, Ben Poole, et al. Reconfusion: 3d reconstruction with diffusion priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21551–21561, 2024.

[176] Daniel Watson, William Chan, Ricardo Martin Brualla, Jonathan Ho, Andrea Tagliasacchi, and Mohammad Norouzi. Novel view synthesis with diffusion models. In *The Eleventh International Conference on Learning Representations*, 2023.

[177] Ruoxi Shi, Hansheng Chen, Zhuoyang Zhang, Minghua Liu, Chao Xu, Xinyue Wei, Linghao Chen, Chong Zeng, and Hao Su. Zero123++: a single image to consistent multi-view diffusion base model. *arXiv preprint arXiv:2310.15110*, 2023.

[178] Jianglong Ye, Peng Wang, Kejie Li, Yichun Shi, and Heng Wang. Consistent-1-to-3: Consistent image to 3d view synthesis via geometry-aware diffusion models. *arXiv preprint arXiv:2310.03020*, 2023.

[179] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[180] Eric R Chan, Koki Nagano, Matthew A Chan, Alexander W Bergman, Jeong Joon Park, Axel Levy, Miika Aittala, Shalini De Mello, Tero Karras, and Gordon Wetzstein. Generative novel view synthesis with 3d-aware diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4217–4229, 2023.

[181] Titas Anciukevicius, Fabian Manhardt, Federico Tombari, and Paul Henderson. Denoising diffusion via image-based rendering. *arXiv preprint arXiv:2402.03445*, 2024.

[182] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[183] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023.

[184] Jiaxiang Tang, Zhaoxi Chen, Xiaokang Chen, Tengfei Wang, Gang Zeng, and Ziwei Liu. Lgm: Large multi-view gaussian model for high-resolution 3d content creation. *arXiv preprint arXiv:2402.05054*, 2024.

[185] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

[186] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak

Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. *Advances in Neural Information Processing Systems*, 36, 2024.

[187] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[188] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[189] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2553–2560. IEEE, 2022.

[190] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[191] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[192] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion GANs. In *International Conference on Learning Representations*, 2022.

[193] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[194] Sadeep Jayasumana, Srikumar Ramalingam, Andreas Veit, Daniel Glasner, Ayan Chakrabarti, and Sanjiv Kumar. Rethinking fid: Towards a better evaluation metric for image generation. *arXiv preprint arXiv:2401.09603*, 2023.

[195] Tuomas Kynkäänniemi, Tero Karras, Miika Aittala, Timo Aila, and Jaakko Lehtinen. The role of imagenet classes in fréchet inception distance. In *The Eleventh International Conference on Learning Representations*, 2023.

[196] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

[197] Toan Tran, Trung Pham, Gustavo Carneiro, Lyle Palmer, and Ian Reid. A bayesian data augmentation approach for learning deep models. *Advances in neural information processing systems*, 30, 2017.

[198] Ruifei He, Shuyang Sun, Xin Yu, Chuhui Xue, Wenqing Zhang, Philip Torr, Song Bai, and XIAOJUAN QI. IS SYNTHETIC DATA FROM GENERATIVE MODELS READY FOR IMAGE RECOGNITION? In *The Eleventh International Conference on Learning Representations*, 2023.

[199] Brandon Trabucco, Kyle Doherty, Max A Gurinas, and Ruslan Salakhutdinov. Effective data augmentation with diffusion models. In *The Twelfth International Conference on Learning Representations*, 2024.

[200] Zineng Tang, Ziyi Yang, Chenguang Zhu, Michael Zeng, and Mohit Bansal. Any-to-any generation via composable diffusion. *Advances in Neural Information Processing Systems*, 36, 2024.

[201] Giuseppe Di Giacomo, Giulio Franzese, Tania Cerquitelli, Carla Fabiana Chiasserini, Pietro Michiardi, et al. Dimvis: Diffusion-based multi-view synthesis. In *ICML 2024 Workshop on Structured Probabilistic Inference & Generative Modeling 2nd SPIGM@ ICML*. ACM, 2024.

[202] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[203] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[204] Wei Xiong, Yutong He, Yixuan Zhang, Wenhan Luo, Lin Ma, and Jiebo Luo. Fine-grained image-to-image transformation towards visual recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5840–5849, 2020.

[205] Fabio Henrique Kiyoiti Dos Santos Tanaka and Claus Aranha. Data augmentation using gans. *arXiv preprint arXiv:1904.09135*, 2019.

[206] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018.

[207] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[208] Yuge Shi, Brooks Paige, Philip Torr, et al. Variational mixture-of-experts autoencoders for multi-modal deep generative models. *Advances in neural information processing systems*, 32, 2019.

[209] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[210] George Stein, Jesse Cresswell, Rasa Hosseinzadeh, Yi Sui, Brendan Ross, Valentin Villecroze, Zhaoyan Liu, Anthony L Caterini, Eric Taylor, and Gabriel Loaiza-Ganem. Exposing flaws of generative model evaluation metrics and their unfair treatment of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

[211] Peichun Li, Hanwen Zhang, Yuan Wu, Liping Qian, Rong Yu, Dusit Niyato, and Xuemin Shen. Filling the missing: Exploring generative ai for enhanced federated learning over heterogeneous mobile edge devices. *IEEE Transactions on Mobile Computing*, 2024.

[212] Xumin Huang, Peichun Li, Hongyang Du, Jiawen Kang, Dusit Niyato, Dong In Kim, and Yuan Wu. Federated learning-empowered ai-generated content in wireless networks. *IEEE Network*, 2024.

[213] Dimitri Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

Appendix A

Proofs of Chapter 4 theorems

In this section, we provide the proofs of the theorems of Chapter 4, which are adapted from the proofs of [145] to align with the different settings of our system.

A.1 Proof of Theorem 1

Theorem 1. *Consider a system of N nodes, all belonging to class 1, with mean and maximum computing capability constraint (resp.) $\bar{\chi}_1$ and $\chi_1^{\max}$. Models have complexity m_1, which implies that d_1 samples and e_1 epochs are required to reach the target learning quality. Also, H=h=1, and if the node receiving the training request accepts, the time required for the training is T_{11}, otherwise a time T_w is waited. Then:*

1. *if all nodes, except for node n, employ GENIAL, then* $\limsup_{t\to\infty} \mu_n^{11}(t) \leq \pi_{11} = \frac{\bar{\chi_1}N}{2} \frac{T_w}{m_1 d_1 e_1 - \frac{\bar{\chi_1}N}{2}(T_{11}-T_w)}$;
2. *if all nodes apply GENIAL, then* $\lim_{t\to\infty} \mu_n^{11}(t) = \pi_{11} = \frac{\bar{\chi_1}N}{2} \frac{T_w}{m_1 d_1 e_1 - \frac{\bar{\chi_1}N}{2}(T_{11}-T_w)}$, $\forall n \in \mathcal{N}$.

Proof. First, since H=h=1, from (4.1) one can derive that Π_{11}=π_{11}, whose value is computed as follows. Using (4.3) and denoting the time required for training as T_{11}, we can derive the mean time required for each session:

$$T_s = T_{11}\Pi_{11} + T_w(1-\Pi_{11}) = \frac{m_1 d_1 e_1}{\chi_1^{max}}\pi_{11} + T_w(1-\pi_{11})\,. \tag{A.1}$$

Being $\frac{1}{N}$ the probability of being selected by the orchestrator as requester, the mean computing cost for a node n when it acts as requester is:

$$c_n^r = \frac{1}{N} m_1 d_1 e_1 \frac{1}{T_s} \pi_{11} \,. \tag{A.2}$$

The mean computing expenditure when the node is a learner c^l is equal to c^r. Thus, imposing the computing constraint it follows that:

$$c_n^{tot} = c_n^r + c_n^l = \frac{2}{N} m_1 d_1 e_1 \frac{1}{T_s} \pi_{11} \leq \bar{\chi_1}. \tag{A.3}$$

Solving equation (A.3) by using (A.1) we obtain the Pareto-optimal value:

$$\pi_{11} = \frac{\bar{\chi_1} N}{2} T_w \frac{1}{m_1 d_1 e_1 - \frac{\bar{\chi_1} N}{2} (T_{11} - T_w)}. \tag{A.4}$$

Consider the scenario where all nodes but u employ GENIAL; a node n, with $n \neq u$, rejects a training request when $\nu_n(t) > \pi_{11}$; therefore, $\limsup_{t\to\infty} \nu_n(t) \leq \pi_{11}, n \neq u$. However, nodes accept a training request regardless of the requester. This means that all nodes receive an equal amount of service, hence $\limsup_{t\to\infty} \mu_n(t) \leq \pi_{11}, n = 1, \ldots, N$. This proves the first part of the theorem.

To prove the second part of the theorem, we start defining the following quantities:

$$\phi_n(S) = \frac{\text{No. of accepted requests made by } n \text{ till } S}{S} \tag{A.5}$$

$$\psi_n(S) = \frac{\text{No. of received sessions accepted by } n \text{ till } S}{S} \tag{A.6}$$

with n being a given node and S denoting the number of total training sessions, both accepted and rejected. Denoting $\phi_n = \lim_{S\to\infty} \phi_n(S)$, since the requester and the learner are selected respectively with probability $\frac{1}{N}$ and $\frac{1}{N-1}$, it is possible to write:

$$\phi_n = \lim_{S\to\infty} \frac{\text{No. of accepted requests made by } n}{S} \cdot \frac{\text{No. of received sessions accepted by } n}{\text{No. of received sessions accepted by } n} = \frac{NRS}{N(N-1)} \tag{A.7}$$

From (A.7) we can see that the NRS converges if and only if ϕ_n converges. Notably, since H=h=1, the probability for a requester that its training request is accepted, hence performed, is precisely equal to the probability that the only node receiving the request accepts, i.e., π_{11}. For this reason, for all nodes, the number of made requests that are accepted must be equal to the total number of accepted received requests. Then, we define the following Lemma.

Lemma 1: $\sum_{n=1}^{N}(\phi_n(S)-\psi_n(S))=0$

Proof. First, we consider the node n at training session S and we prove that $\phi_n(S)-\psi_n(S)$ converges to 0.

We define now the following equations:

$$\phi_n(S+1) = \frac{S\phi_n(S) + 1_A}{S+1}$$
$$\psi_n(S+1) = \frac{S\psi_n(S) + 1_R}{S+1}$$

where

$$1_A = \begin{cases} 1, \text{if request made by node } n\text{' is accepted} \\ 0, \text{else} \end{cases}$$

$$1_R = \begin{cases} 1, \text{if node } n \text{ accepts a received training request} \\ 0, \text{else.} \end{cases}$$

Then, we define

$$\rho(S) = [\phi_1(S) - \psi_1(S), \ldots, \phi_N(S) - \psi_N(S)]^T \tag{A.8}$$

which allows us to write:

$$\rho(S+1) = \rho(S) + \frac{1}{S+1}(-\rho(S) + \omega(S)) \tag{A.9}$$

with $\omega(S)$ defined as follows:

$$\omega(S) = \begin{cases} -1, \text{if node } n \text{ accepts a received training request;} \\ 1, \text{if request made by node } n \text{ is accepted;} \\ 0, \text{else.} \end{cases}$$

The objective is to prove that the sequence $\{\rho_S\}$ converges to $\rho^*=[0,\ldots,0]^T$ when using GENIAL, as this leads to the proof that $\phi_n(S)-\psi_n(S)$, and hence $\nu_n(S)-\mu_n(S)$, converges to 0. To prove this, we utilize the following corollary [213]. □

Corollary 1: We define the following sequence $\{\theta(S)\}$:

$$\begin{aligned} &\theta(S+1) = \theta(S) + \lambda(S)\sigma(\omega(S), \theta(S)) \\ &\sum_{S=1}^{\infty} \lambda(S) = \infty \\ &\sum_{S=1}^{\infty} \lambda^2(S) < \infty \end{aligned} \tag{A.10}$$

Setting $\bar{\sigma}(\theta)=E[\sigma(\theta,\omega)]$, if:
a) $(\theta^*-\theta)^T \bar{\sigma}(\theta) \geq C_1 \|\theta^\star-\theta\|^2$ for some $C_1>0$;
b) $E\left[\|\sigma(\theta,\omega)\|^2\right] \leq C_2 \left[\|\theta^\star-\theta\|^2+1\right]$ for some $C_2 > 0$;
it follows that $\lim_{S\to\infty} \theta(S)=\theta^\star$.

We must prove that $\rho(S)$ converges to $\rho^\star$. With $\lambda(S)=1/(S+1)$ and $\sigma(\theta,\omega) = -\rho(S)+\omega(S)$, we verify that (A.9) satisfies (A.10); also, condition b) of Corollary 1 is verified considering sufficiently large C_2. Now, we prove that condition a) is satisfied, i.e.,

$$(-\rho)^T \bar{\sigma}(\rho) \geq C_1 \| -\rho\|^2. \tag{A.11}$$

At training session S, if we have a set $\mathcal{A}$ of A nodes out of N that are accepting the training requests – with the remaining $N-A$ nodes that rejects the requests – it holds that $\mu_n(S)-\nu_n(S)>\delta, n=1,\ldots,A$, and $\mu_n(S)-\nu_n(S)<\delta$, $n=A+1,\ldots,N$. Therefore, for some $\epsilon>0$, $\phi_n(S)-\psi_n(S)>\epsilon$, $n=1,\ldots,A$ and $\phi_n(S)-\psi_n(S)<\epsilon$, $n=A+1,\ldots,N$.

Considering that a node becomes a requester with probability equal to $1/N$, the probability that a node in $\mathcal{A}$ makes a request that is accepted is equal to $(A-1)/[N(N-1)]$. On the other hand, the node receives and accepts a training request

with probability $(N-1)/[N(N-1)]$, while $A/[N(N-1)]$ defines the probability that a node in the set of the rejecting ones makes a training request and that the latter is accepted. Finally, the probability of accepting a request is 0. Therefore, we can write the following:

$$\bar{\sigma}_n(\rho(S),\omega(S)) = \begin{cases} -\rho_n(S) + \frac{A-N}{N(N-1)}, & \text{if } n = 1,\ldots,A \\ -\rho_n(S) + \frac{A}{N(N-1)}, & \text{if } n = A+1,\ldots,N \end{cases}$$

Thus, we derive that

$$\begin{aligned} \left(-\rho_S\right)^T \bar{\sigma}\left(\rho_S\right) =& \|\rho(S)\|^2 - \frac{1}{N(N-1)}\sum_{n=1}^{A}(A-N)\rho_n(S) - \frac{1}{N(N-1)}\sum_{n=A+1}^{N} A\rho_n(S) \\ =& \|\rho(S)\|^2 + \frac{1}{N-1}\sum_{n=1}^{A}\rho_n(S) \\ >& \|\rho(S)\|^2 + \frac{A}{N-1}\epsilon \\ >& \|\rho(S)\|^2. \end{aligned}$$

Thus, a) holds with C_1=1 and we can use Corollary 1. Since $\lim_{S\to\infty}\rho(S)=\rho^*$ with probability 1, it follows that $\phi_n(S)-\psi_n(S)$ and $\mu_n(S)-\nu_n(S)$ as well converge to 0 for each n.

For a node n that uses the GENIAL algorithm, it holds that $\limsup_{S\to\infty}\nu_n(S)\leq\pi_{11}$. Also, if $\nu_n(S)\leq\pi_{11}$ and $\nu_n(S)-\mu_n(S)=0$, the node n always accepts a training request, meaning that $\liminf_{S\to\infty}\nu_n(S)\not<\pi_{11}$. It follows that $\lim_{S\to\infty}\nu_n(S)=\pi_{11}$ and hence $\lim_{S\to\infty}\mu_n(S)=\pi_{11}$ as well, as $\nu_n(S)-\mu_n(S)$ converge to zero.

If both $\nu_n(S)$ and $\mu_n(S)$ converge to π_{11} considering the number of training sessions, they converge also in time, thus we can write $\lim_{t\to\infty}\nu_n(t)=\lim_{t\to\infty}\mu_n(t)=\pi_{11}=\frac{\bar{\chi}_1 N}{2}T_w\frac{1}{m_1 d_1 e_1-\frac{\bar{\chi}_1 N}{2}(T_{11}-T_\omega)}$. □

A.2 Proof of Theorem 2

Theorem 2. *Consider a system of N nodes and K classes, H=h=1, and N_k nodes in class k. Also, the computing capabilities constraints are as follows: $\bar{\chi}_1<\bar{\chi}_2<\ldots<\bar{\chi}_K$ and $\chi_1^{\max}<\chi_2^{\max}<\ldots<\chi_K^{\max}$. Then:*

1. *if all nodes, except for node n, employ GENIAL, then* $\limsup_{t\to\infty} \mu_n^{kj}(t) \leq \pi_{kj}$;
2. *if all nodes apply GENIAL, then* $\lim_{t\to\infty} \mu_n^{kj}(t) = \pi_{kj}$, $\forall n \in \mathcal{N}$ *and* $k, j = 1, \ldots, K$.

Proof. In sessions of type (kk), all nodes behave as if they were of the same class, i.e., as they had the same computing constraints $\bar{\chi}_k$ and $\chi_k^{\max}$. Thus, using Theorem 2 we can prove that $\nu_n^{kk}(t)$ and $\mu_n^{kk}(t)$ will converge. Then, as (4.8) and (4.9) allow us to derive the ratio $\frac{\Pi_{kj}}{\Pi_{kk}}$ and, hence, $\frac{\pi_{kj}}{\pi_{kk}}$, we can prove the convergence for each session type. □

A.3 Proof of Theorem 3

Theorem 3. *Consider a system with N nodes, K classes, $H>1$, $h>1$ and N_k nodes in class k, $k=1, \ldots, K$. As for the computing capabilities constraints, assume that* $\bar{\chi}_1 < \bar{\chi}_2 < \ldots < \bar{\chi}_K$ *and* $\chi_1^{\max} < \chi_2^{\max} < \ldots < \chi_K^{\max}$. *Then:*

1. *if all nodes, except for node n, use GENIAL,* $\limsup_{t\to\infty} \mu_n^{kj}(t) \leq \Pi_{kj}$;
2. *if all nodes apply GENIAL, then* $\lim_{t\to\infty} \mu_n^{kj}(t) = \Pi_{kj}$, $\forall n \in \mathcal{N}$ *and* $k, j = 1, \ldots, K$.

Proof. In the general case, we have that $H>1$ and $h>1$; thus, fixing such quantities, we can prove that $\nu_n^{kk}(t)$ and $\mu_n^{kk}(t)$ converge, by following the same procedure of the proof of Theorem 2 and by appropriately scaling Lemma 1. □