



POLITECNICO
MILANO 1863

Department of Electronics, Information and Bioengineering
Doctoral Programme In Information Technology

Machine Learning as a Network Management Primitive: From End-to-End Optimization to Atomic Network Functions

Doctoral Dissertation of:
Nicola Di Cicco

Advisor: Prof. Massimo Tornatore
Tutor: Prof. Ilario Filippini
Year 2023/2024 - XXXVII Cycle

Abstract

Machine Learning (ML) is rapidly becoming the jack-of-all-trades of the network management stack. Thanks to its purely data-driven nature, ML can be leveraged for developing a broad set of network management functions tailored for the specific task at hand. Unfortunately, we are still far from harnessing the full potential of ML for network management. Contemporary literature in this field has made tremendous progress in identifying algorithms and design principles for producing well-trained models from historical data. These contributions, though fundamental, cover only one aspect of the whole ML model lifecycle. Deploying, using, and maintaining ML-based network management solutions poses nontrivial research challenges, orthogonal with respect to training a model, that is presently not yet thoroughly addressed. This hinders a widespread adoption of ML by network operators, who instead prefer relying upon classical, battle-tested network management solutions. To take a step toward solving this fundamental problem, this Thesis identifies five core challenges in ML for network management: 1) Generalizability to data beyond training, 2) Adaptability to dynamic network environments, 3) Reliability, in terms of providing theoretical performance guarantees after model deployment, 4) Data efficiency, for minimizing the amount of labor required for training and updating models, and 5) Performance, i.e., ML-based solutions must tangibly improve over conventional methods to be worth considering. We address these challenges through multiple representative network management applications: online and offline network optimization with Reinforcement Learning, focusing on generalizability and performance; ML-based hardware fault classification in microwave networks, focusing on data efficiency and reliability; Continual in-network ML, focusing on data efficiency and adaptability; and intent-based networking with Large Language Models, focusing on performance and data-efficiency. We quantitatively validate the practical effectiveness of our proposed solutions through extensive comparisons against the state of the art, and by leveraging novel real-world datasets, which we make publicly available.

Keywords: Machine Learning; Network Management; Optimization

Contents

Abstract	i
Contents	iii
1 Introduction	1
1.1 Problem Statement	2
1.2 Thesis Structure	5
1.3 Other Contributions	14
2 State of the Art	19
2.1 Machine Learning for Network Optimization	19
2.2 Machine Learning for Failure Identification	22
2.3 Machine Learning for Service Orchestration	26
2.4 In-Network Machine Learning	28
2.5 Language Models for Network Automation	29
3 Reinforcement Learning for Network Optimization	31
3.1 On Reinforcement Learning for Routing and Wavelength Assignment . . .	31
3.2 Reinforcement-Learning-based Local Search for Network Optimization . . .	45
4 Data-Centric and Reliable Machine Learning for Fault Management	59
4.1 Data-Centric Machine Learning for Hardware Fault Classification	59
4.2 As-Soon-As-Possible Hardware Fault Classification with Guarantees	73
5 Machine Learning for Service Orchestration	85
5.1 Scalable Service Orchestration with Reinforcement Learning	85
5.2 Reinforcement Learning for Multi-Objective Service Orchestration	96
6 Continual In-Network Learning	109

7	Large Language Models for Automated Network Configuration	113
8	Conclusion and future developments	119
8.1	Conclusion	119
8.2	Future developments	121
	Bibliography	123
	List of Acronyms	143
	Copyright Statement	145

1 | Introduction

Machine Learning (ML) is rapidly becoming a central technology in future communications networks [1, 2], with applications including (but not limited to) traffic engineering [3–6], traffic classification [7–10], traffic prediction [11–13], intelligent transport protocols [14–17], fault management [18–20], physical layer modeling [21–23], and the list goes on. The promise of ML for networks is an ambitious one: by leveraging the abundant amount of data produced by modern communication networks, we can automate a broad set of tasks that require heavy human intervention and specialized domain knowledge.

Unfortunately, despite the extensive research efforts concentrated in the past few years, we are still far from harnessing the full potential of ML for network management. Contemporary literature in ML for network management has extensively investigated algorithms and design principles for producing well-trained models from historical data and/or experiences, e.g., identifying which ML methodologies are the more effective for attacking specific network management problems, or which features are the most important for solving a certain task [1, 24, 25]. These contributions, while fundamental, comprise only a first step towards a complete production pipeline. Deploying, using, and maintaining ML-based network management solutions poses nontrivial research and engineering challenges, orthogonal with respect to training a ML model, that are presently not yet thoroughly addressed. This hinders widespread adoption of ML by network operators, who prefer relying on classical, battle-tested network management solutions.

To take a step towards solving this fundamental problem, this Thesis identifies five major open challenges in ML for network management and proposes tangible, applicable solutions for solving them across a set of representative application domains, namely, *i*) network optimization, *ii*) service orchestration, *iii*) fault management, *iv*) traffic classification, and *v*) zero-touch networking. This broad set of applications requires different, specialized ML tools, ranging from “classical” models (e.g., ensembles of decision trees) to ad-hoc neural network architectures, from conventional Supervised Learning to heavily customized Reinforcement Learning, and from standard offline training to ML-driven data collection and augmentation. As we will discuss in detail throughout the whole

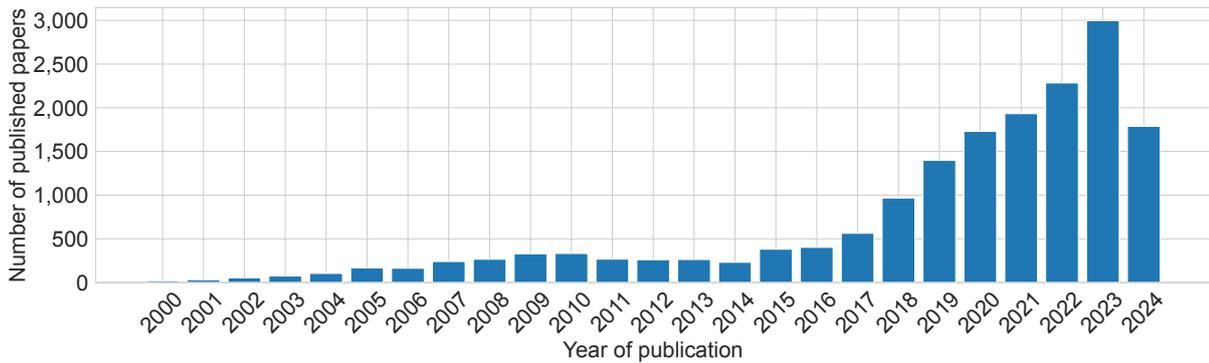


Figure 1.1: Approximate number, as of October 2024, of published papers over the years on IEEE Xplore on Machine Learning applied to communications networks.

manuscript, each of these methodologies will be instrumental in targeting (at least) one of the aforementioned core challenges.

This introductory Chapter first describes the general research problem of ML applied to network management and details the main research challenges that shall be addressed throughout this work. We then overview the content of the remainder of this Thesis, placing each contribution in the context of the aforementioned research challenges, and anticipating our main results and takeaways.

Overall, the main goal in writing this document is to provide a fresh perspective about what it means, in practice, to apply ML for network management, raising awareness on the challenges beyond “just” fitting a ML model to solve a specific task. We hope this will prove useful to the interested reader, and will inspire future work in this thriving field.

1.1. Problem Statement

As anticipated before, ML has received significant attention in the networking community, especially in the past few years. As a rough attempt to quantitatively capture this trend, Fig. 1.1 illustrates the result of a query run over the years on the IEEE Xplore database, searching for the co-occurrence of indexing terms such as “Artificial Intelligence”, “Machine Learning” together with “Communications networks”, “Computer networks” and so on. We can observe that the number of published papers combining ML and networking started to rapidly increase after approximately the year 2016; indeed, the majority of published papers on this topic are concentrated in the past few years! We speculate this is because, approximately starting from the years 2015-2016, software tooling for building and training ML models was becoming stable and accessible enough for non-ML-specialists to experiment with. As notable examples, libraries such as Tensorflow [26], PyTorch [27],

and scikit-learn [28] succeeded in abstracting away complex implementation details of ML models and in providing intuitive programming interfaces in Python, which is a simple (as opposed to, e.g., C++) and high-level scripting language. This has allowed the networking community at large to easily prototype and experiment ML-based solutions for a broad set of novel datasets and problems. This line of investigation successfully demonstrated that ML models can effectively learn complex statistical patterns from networking data, and achieve excellent performance in a variety of predictive tasks.

However, ML-based approaches to network management, as of today, are not widely deployed in real-world networks. Indeed, and this was confirmed to us by multiple industrial partners we cooperate with, network operators prefer to engineer their traffic with classical optimization algorithms, classify their network faults with knowledge-driven rule-based systems, model their physical layer with simple parametric formulas, and so on. This suggests that simply demonstrating that ML-based approaches can be successfully applied to network management tasks is a *necessary, but not sufficient requirement* for making them worth switching over consolidated methodologies. Instead, there must exist a set of practical challenges, orthogonal to training a model, that have not yet been thoroughly addressed by contemporary literature. Given this premise, the central research question encompassing all of this manuscript can be summarized as follows:

What are the main challenges currently preventing a widespread application of ML for network management, and how do we practically solve them?

To take a step toward solving this fundamental question, through an extensive review of recent literature and continuous interaction with industrial partners, this Thesis identifies the following five research challenges.

Challenge 1: Generalizability. One basic property of well-trained ML models is their capability of generalizing their learned logic to data outside the training set. While this seems trivial, in several network management applications (e.g., routing, traffic engineering), a ML model might not be straightforwardly applied to data “significantly different” than training. Consider, for example, applications in which the properties of the input data are intrinsically tied to the characteristics of the considered network topology, such as ML for network optimization and control. In such applications, it is extremely desirable to, e.g., train the ML model in a small, easy-to-simulate network and then generalize its learned logic to larger, more practical networks. However, conventional ML models are restricted to input and output size of fixed dimensionality, which makes it challenging to achieve this goal. For this reason, there is a need for ML-based approaches that address generalizability by design.

Challenge 2: Adaptability. Despite whichever generalization capabilities, deployed ML models will require, at some point, to be re-trained or fine-tuned on new data. This is because, eventually, the statistical patterns captured by the training set will no longer reflect fresh observed data, resulting in tangible performance degradation over time. Such phenomenon, known in ML literature as *concept drift*, is ubiquitous in network management. As a motivating example, consider a simple binary traffic classification task, where a ML model has to decide whether a flow is either benign or malicious. As the users' behavior and the cybersecurity landscape rapidly evolve over time, so do the statistical properties of the traffic observed by the ML model. For this reason, deployed models need to be constantly monitored and periodically updated with fresh data. This adds multiple layers of complexity to ML model deployment, e.g., when and how to retrain, and which data should be used for retraining.

Challenge 3: Data Efficiency. It is well-known that ML thrives in the presence of abundant training data. Indeed, the most straightforward solution for improving the performance of a given ML model is to increase the training set in tandem with the model size [29, 30]. However, constructing large training datasets is a major challenge in several important network management applications, such as fault classification and Intent-Based Networking (IBN). Consider, for example, ML-based classification of network faults, a fundamental component of zero-touch fault management. For such an application, good-quality data is neither abundant nor easy to curate, since it requires extensive domain knowledge to interpret and annotate. A similar argument can be made for IBN, for which a training dataset must comprise pairs of, e.g., natural-language intents and the domain-specific equipment configuration realizing those intents. Because of this, network operators incur substantial costs for training and updating ML models. It is, therefore, of paramount importance to devise ad-hoc data augmentation and annotation strategies for training top-performing ML models with the least amount of data possible.

Challenge 4: Reliability. ML models are complex nonlinear functions fit to high-dimensional data distributions. Even though their efficacy can be validated through extensive performance testing, how ML model will respond to data outside the training set (i.e., generalize) is currently poorly understood. In a worst-case scenario, a ML model might perform arbitrarily bad on test data. In mission-critical network management applications, e.g., fault detection and remediation, uncontrollably bad predictions can result in catastrophic outcomes. Moreover, state-of-the-art ML models such as Gradient-Boosted Decision Trees and Deep Neural Networks tend to overestimate the confidence in their prediction [31, 32], thus instilling in the user a misleading sense of security. In summary, because of their inherent lack of performance guarantees, it is difficult for network oper-

ators to delegate mission-critical responsibilities to ML-based solutions. Because of this, there is a pressing need in network management for *reliable* ML methods able to guarantee user-provided performance constraints during deployment (e.g., equipment fault predictions must be correct with high probability).

Challenge 5: Performance. ML-based solutions for network management must compete against conventional approaches, which are backed up by decades of literature and have a well-established industrial adoption. One prominent example is network optimization, which builds upon sophisticated solving methods from Operations Research (especially Linear and Integer Linear Programming). Indeed, despite the ambitious expectations put forward and the considerable investments from both academia and industry, ML for network management has yet to experience its paradigm-shifting “ImageNet moment”.¹ It is, therefore, of paramount importance to quantitatively demonstrate tangible advantages compared to consolidated methodologies.

In this Thesis, starting from major use cases for ML in network management (e.g., optimization, fault management), we identify which challenge is the most critical for a specific application, and we propose novel methodologies for dealing with such challenge. Though the developed methodologies are tailored to our considered use cases, we expect the general design principles to be transferable across different domains.

1.2. Thesis Structure

This Thesis, in accordance with Politecnico di Milano’s rules and regulations, is presented as a collection of articles, comprising both published and unpublished material. For each article presented in this Thesis, I made leading author contributions in terms of conceptualization, software development, and paper writing. Each central Chapter presents one or two articles, grouped by topic and introduced by a short foreword.

We now briefly overview the content of the remainder Chapters of this Thesis. Specifically, for each technical Chapter, we introduce the specific challenges addressed by the presented contributions in the scope of the Thesis work, and we anticipate their central results.

Chapter 2: State of the Art. This Chapter extensively reviews the literature on ML applied to network management in the context of the specific topics investigated in

¹In 2012, the AlexNet paper [33] achieved a breakthrough result in the large-scale image recognition competition ImageNet, outclassing classical Computer Vision (CV) approaches. In contrast to conventional solutions primarily based on handcrafted image processing algorithms, AlexNet adopted a pure ML approach based on Deep Convolutional Neural Networks. These results fundamentally changed the direction of the whole CV community, which is now almost exclusively focused on ML-based approaches, and of Artificial Intelligence at large, which is presently dominated by extremely large neural networks.

this Thesis. The purpose of this Chapter is both to get the reader acquainted with the general topics addressed in this Thesis, and to highlight the novel contributions of this work compared to the current state of the art.

Chapter 3: Reinforcement Learning for Network Optimization. This Chapter investigates using ML, in particular Reinforcement Learning (RL), for learning solving algorithms for combinatorial network optimization problems. The main challenges addressed in this Chapter relate to performance, since we want our RL-based algorithms to be competitive with classical Operations Research (OR) algorithms, and on generalization, since we expect RL-based algorithms to work well on problem instances different (and possibly larger) than training. Following this line of investigation, this Chapter presents the following research papers:

- [Nicola Di Cicco](#), Emre Furkan Mercan, Oleg Karandin, Omran Ayoub, Sebastian Troia, Francesco Musumeci, Massimo Tornatore. On Deep Reinforcement Learning for Static Routing and Wavelength Assignment, *IEEE Journal of Selected Topics in Quantum Electronics*, 2022 [34]
- [Nicola Di Cicco](#), Memedhe Ibrahim, Sebastian Troia, Massimo Tornatore. DeepLS: Local Search for Network Optimization Based on Lightweight Deep Reinforcement Learning, *IEEE Transactions on Network and Service Management*, 2023 [35]

Optimization is one of the cornerstones of modern network management, especially in terms of modeling and solving hard combinatorial problems. For this reason, novel scalable and efficient solving algorithms are always sought. The core contribution of this Chapter lies in the design of generalizable learned optimization algorithms based on RL, and in an extensive evaluation of their effectiveness against state-of-the-art metaheuristics.

The motivation for applying ML to (network) optimization can be summarized as follows: real-world optimization problem instances are often statistically correlated. As a motivating example, suppose we want to solve the same traffic engineering problem every 24 hours. It is reasonable to assume that the traffic matrices in two consecutive instances will be similar, and hence, the corresponding optimal routings will also be similar (or at least, not radically different). As another example, consider the Routing and Wavelength Assignment problem in optical networks. Though this problem is NP-Hard for general networks, real-world optical-network topologies do not have an arbitrary structure. Indeed, optical network topologies are generally planar graphs with low nodal degrees and high diameters, and commonly used heuristics take advantage of this property (e.g., by filling longer paths first). In other words, real-world problem data exhibits a statistical structure that can be exploited for designing specialized and efficient solving algorithms.

However, manually designing generalized algorithms that capture and leverage a problem’s structure is extremely challenging. Instead, it would be much more desirable if we could *learn* these properties from data and autonomously exploit them with minimal human input. Indeed, the central result of this Chapter is *not* to claim state of the art in solving a specific problem, but that RL-based methods can autonomously learn algorithms at least as good as sophisticated metaheuristics, which require considerable domain-expert knowledge and implementation effort.

The contributions in this Chapter leverage RL to learn algorithms for network optimization. RL is among the most prominent paradigms for learning how to solve network optimization algorithms. This is because RL is a flexible and general method for attacking any sequential decision-making problem, requiring only the specification of a user-specified goal (i.e., a “reward function”).

The first paper presented in this Chapter leverages RL for learning a *constructive heuristic* for Routing and Wavelength Assignment (RWA) problems. This algorithm can be regarded as an “End-to-End” method since, given a RWA problem definition, produces a single feasible solution for the input problem. The design of the procedure is simple: for each demand, the RL agent (implemented as a neural network) decides which path and which wavelength to assign based on the network’s state. This is akin to commonly-used heuristics for RWA, e.g., K-Shortest-Paths First-Fit, but leveraging learned decision rules as opposed to a static heuristic. The core contribution of this work lies in the extensive numerical comparisons against a state-of-the-art metaheuristic, namely, a heavily customized Genetic Algorithm at the core of countless publications in my research group. There are two main conclusions from this work: *i)* RL-based algorithms can generally outperform standard greedy heuristics, *iii)* RL-based algorithms, though being significantly faster than state-of-the-art metaheuristics, fall short in terms of solution quality; *iii)* Conventional RL approaches based on Multi-Layer Perceptrons (MLPs) neural networks cannot trivially be applied to network topologies different from training. This is because the inputs and outputs of MLPs are vectors whose dimensionality cannot be changed at inference. Performance and, in particular, generalizability to multiple network topologies, will be extensively addressed in the follow-up work presented in this Chapter.

Having established the limitations of end-to-end RL-based approaches, with DeepLS we propose a different approach. Our idea is based on the observation that stochastic local search algorithms are the state of the art for many combinatorial optimization problems. For this reason, instead of attempting to learn an end-to-end algorithm, we delegate to RL one of the most critical choices in local search: which neighbors to explore, based on the current solution state. As mentioned before, fundamental desiderata for RL-based algo-

rithm are generalization capabilities to network topologies different (and possibly larger) than training. We achieve this objective by leveraging a special class of neural networks, namely, equivariant neural networks, which enable input and output dimensionality of variable size at inference. Remarkably, DeepLS was able not only to outperform end-to-end RL-based algorithms, but also domain-informed metaheuristics for two challenging network optimization problems, namely, OSPF Traffic Engineering and RWA. Moreover, DeepLS could generalize its learned decision rules without retraining to network topologies 5x larger than training. The main takeaway of this work is as follows: we can achieve great performance benefits by taking a well-known algorithmic framework and by substituting certain specific subroutines with ML models tailored for the problem at hand. Our results are promising, and we expect this design principle to become a standardized approach for future research on ML-based network optimization.

Code and data for all the contributions presented in this Chapter are publicly available.²

Chapter 4: ML for Fault Management. This Chapter investigates using ML-based methods for identifying hardware failures in microwave network equipment. The challenges addressed here relate to data efficiency, since data annotation for failure classification is extremely expensive, and reliability, since fault classification is a mission-critical task. To this end, this Chapter presents the following research papers:

- [Nicola Di Cicco](#), Memedhe Ibrahim, Francesco Musumeci, Federica Bruschetta, Michele Milano, Claudio Passera, Massimo Tornatore. Machine Learning for Failure Management in Microwave Networks: A Data-Centric Approach, *IEEE Transactions on Network and Service Management*, 2024 [36]
- [Nicola Di Cicco](#), Memedhe Ibrahim, Omran Ayoub, Federica Bruschetta, Michele Milano, Claudio Passera, and Francesco Musumeci. ASAP Hardware Failure-Cause Identification in Microwave Networks using Venn-Abers Predictors, *IEEE Transactions on Network and Service Management*, 2024 [37]

Both papers present significant contributions in the scope of a cooperative project carried out with SIAE Microelettronica, an Italian company leader in wireless communications technology. The main purpose of this project was to design ML-based methods for automated classification of failures in microwave radios based on a set of equipment alarm signals. The use of ML is motivated by the large number of alarm signals generated by a single radio equipment (more than 150 every second, aggregated over fifteen minutes), which makes human inspection of each individual record infeasible.

²<https://github.com/bonsai-lab-polimi/tnsm2023-deepls>

It quickly became abundantly clear that the actual challenge of this project was not to devise advanced ML models for fault classification in microwave equipment, but to construct a *good-enough* training dataset for ML model training, and to provide *performance guarantees* once said model was deployed in production. This was in stark contrast with related literature in the field, which mainly focuses on the design of ML models and on improving the model’s performance on a set of benchmarks.

The work on Data-Centric ML, specifically, deals with the problems of automated data augmentation and data collection. Unlike other popular ML applications like image classification or natural language processing, curating a dataset for the classification of faults in microwave equipment is extremely expensive and time-consuming. This is because data curation and annotation must be performed by domain experts familiar with the specific microwave equipment, i.e., commonly-used cheap crowdsourcing services (e.g., Amazon Mechanical Turk [38]) are not an option. This work proposes two fundamental solutions: using *generative models* for producing synthetic data at training time to re-balance unrepresented failure classes, and using *Active Learning* at deployment time to guide the data annotation procedure. The quantitative gains of our methodology applied to a production dataset are significant: we can attain state-of-the-art performance with 4.5% less annotated data, and improve by up to 2.5% the classification’s F1-Score for the least-represented failure class. Similar gains were communicated to us when the solution was deployed in production. Last but not least, all the data used in this study has been made publicly available: to the best of my knowledge, our is the first public dataset of this kind, and my hope is that it will become a standard benchmark for future research in this field.³

The work on As-Soon-As-Possible (ASAP) failure classification deals with the problem of producing fault predictions as fast as possible while, at the same time, providing performance guarantees. The problem statement of this work was motivated by how ML-based fault classification in microwave networks is performed in practice. Microwave radio equipment emits alarm signals at a fixed, fine time granularity (e.g., 1 second). The standard approach for ML-based fault classification is to extract features from a fixed-size temporal window of telemetry (e.g., 15 minutes), thus introducing a significant delay between the occurrence of a fault and its classification. To solve this problem, we proposed an inference strategy for deployed ML models that performs stream-processing of the data at the 1-second granularity, and outputs a prediction as soon as the probability of correct classification exceeds a user-specified threshold. To achieve this goal, we propose using Venn-Abers Predictors [39], a post-training algorithm from the field of Conformal Predict-

³<https://github.com/bonsai-lab-polimi/tnsm2024-data-centric>

tion, which can turn any trained ML models into a probabilistic classifier with rigorous statistical guarantees. On a production dataset, the prediction of our system can satisfy constraints on the probability of correct classification as tight as 99%. As for the previous paper, the dataset used in this study, extracted from real-world microwave networks and manually annotated by domain experts from the industry, is publicly available.⁴

Chapter 5: ML for Service Orchestration. This Chapter investigates using RL for learning how to dynamically and optimally orchestrate services over a network of heterogeneous devices. Similarly to Chapter 3, the main challenges considered here relate to generalization to larger network topology, and improved performance compared to classical methods. However, while before we considered *offline* network optimization, we now consider an *online* resource allocation, where demands enter the network and free resources dynamically according to unknown statistical distributions. In this context, our main technical contributions are presented in the following research papers:

- Nicola Di Cicco, Gaetano Francesco Pittalà, Gianluca Davoli, Davide Borsatti, Walter Cerroni, Carla Raffaelli, Massimo Tornatore. DRL-FORCH: A Scalable Deep Reinforcement Learning-based Fog Computing Orchestrator, *IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023. [40]
- Nicola Di Cicco, Gaetano Francesco Pittalà, Gianluca Davoli, Davide Borsatti, Walter Cerroni, Carla Raffaelli, Massimo Tornatore. Scalable and Energy-Efficient Service Orchestration in the Edge-Cloud Continuum with Multi-Objective Reinforcement Learning, *under review*.

As anticipated, we now consider the problem of assigning services to heterogeneous hardware resources in the Edge-Cloud Continuum [41]. In this scenario, incoming service requests are allocated to hardware resources with vastly different characteristics in terms of computing power, storage, round-trip latency, and energy consumption. As such, from the point of view of the network operator, the service orchestrator must not only aim at maximizing the network’s resource utilization, but also take into account the requests’ Quality-of-Service (QoS) and the overall network’s energy consumption. This translates into a complex online optimization problem.

In DRL-FORCH, we extend the FORCH [42] service orchestrator by substituting its heuristic service allocation policy with a RL-based one, with the goal of maximizing the service acceptance probability while minimizing QoS violations. The main challenge of this work is generalizability: as the available hardware resources in the Edge-Cloud Continuum can change over time, we need to learn a RL-based policy that can operate under

⁴<https://github.com/bonsai-lab-polimi/tnsm2024-asap-venn-abers>

a variable number of possible network nodes, possibly significantly larger than training. We achieve this goal by proposing a novel RL policy architecture based on Deep Sets [43], which extends and improves the model architecture previously in DeepLS. Numerical results are promising: not only our RL-based approach outperforms conventional resource allocation heuristics, but the model maintains its performance for networks up to two orders of magnitude larger than training. Moreover, since Deep Sets process each input independently, they can take advantage of massive parallelization, resulting in short inference times ($< 1\text{ms}$) even on very large networks.

In the journal extension of DRL-FORCH, which is presently under review, we extend the scope of the orchestrator’s goals to optimize the network’s energy consumption, and we adopt a proper Multi-Objective Optimization approach to the resulting optimization problem. In the literature, when dealing with multiple competing objectives, it is common practice to merge them into a single utility function and treat the optimization problem as if it were single-objective. While this solution is viable, it requires identifying a priori how to combine the objectives, e.g., via weighted linear combinations. Instead, in our paper, we propose to derive the entire Pareto Front of RL policies, i.e., a set of policies achieving an “optimal” trade-off between the objectives. This approach separates optimization from decision-making, i.e., it allows network operators to analyze a posteriori how competing objectives interact, and pick the trade-off that best matches their operational requirements. To perform a realistic performance evaluation, we profile the power consumption as a function of the CPU load of multiple edge devices, alongside the hardware resource utilization of popular cloud services. Our results illustrate that our RL algorithm discovers Pareto Fronts on average 30% better than heuristic baselines and is able to generalize their performance to networks several times larger than training, further reinforcing the main empirical conclusions put forward in our prior conference paper.

Code and data for DRL-FORCH are publicly available,⁵ while code and data for the journal extension will be made available once the manuscript is accepted.

Chapter 6: Continual In-Network Learning. This Chapter considers the problem of lifelong adaptation of deployed in-network ML models to streaming data. Here, we address adaptability together with data efficiency, as we want to re-adapt our models with as few labeled data as possible. We address these challenges in the following poster:

- Nicola Di Cicco, Amir Al Sadi, Chiara Grasselli, Andrea Melis, Gianni Antichi, Massimo Tornatore. Poster: Continual Network Learning, *Proceedings of the ACM SIGCOMM Conference*, 2023 [44]

⁵<https://github.com/bonsai-lab-polimi/netsoft2023-drl-forch>

One fundamental truth of all ML applications is that training data will eventually become outdated, and networking is no exception to this rule. Because of this, the performance of a trained ML model will degrade over time. This phenomenon, known in the literature as “concept drift,” formally relates to the evolution over time of the statistical properties of the data. As a motivating example, in the paper, we assess the presence of concept drift in backbone network traffic, and we quantitatively evaluate its negative impact on ad-hoc ML tasks. We then consider a scenario where ML models are programmed into network equipment (e.g., P4 switches). Our goal is twofold: *i*) we want to adapt to novel patterns in the input data while avoiding discarding past learned knowledge (a phenomenon known as “catastrophic forgetting” in ML literature), and *ii*) we want to achieve adaptation with the minimal amount of annotated data possible, since annotating network traffic is expensive and time-consuming. To achieve these goals, we propose an architecture integrating Continual Learning for seamless adaptation without forgetting, and Active Learning for data-efficient retraining. Specifically, the in-network ML model’s predictions are used not only to classify incoming traffic, but also to selectively filter “novel” data to be forwarded to the control plane. Preliminary results on a representative in-network attack classification use-case are promising, with accuracies close to an oracle baseline while requiring annotating less than 2% of a five-day data stream. Our results are limited to simulations, leaving ample room for future research on model deployment onto programmable switches and on performance evaluation on multiple datasets and use cases beyond traffic classification.

Chapter 7: Large Language Models for Network Automation. This Chapter investigates using Large Language Models (LLMs) for designing expert systems for network management. Specifically, we consider the problem of automated network configurations, i.e., translating natural-language user instructions into a domain-specific language for programming network components. The main challenges considered in this Chapter relate to data efficiency, as our goal is to leverage large pre-trained models and adapt them to a completely novel task with the least amount of data possible, and to performance, as automated network configuration must be highly accurate to be useful to their end-user. In this context, our technical contributions are presented in the following paper:

- Nicola Di Cicco, Memedhe Ibrahimi, Sebastian Troia, Francesco Musumeci, Massimo Tornatore. Open Implementation of a Large Language Model Pipeline for Automated Configuration of Software-Defined Optical Networks, in 50th European Conference on Optical Communications (ECOC), 2024 [45]

LLMs have revolutionized the field of Natural Language Processing (NLP) and of Artificial Intelligence (AI) as a whole. Trained on massive corpora of textual data, LLMs are gener-

alist models able to quickly adapt to novel tasks by simply *prompting*, i.e., by instructing them in natural language. This singular phenomenon, known in the literature as “In-Context Learning”, is a capability that emerges as the model’s scale becomes very large [46, 47]. I assume the reader has used a chatbot service at least once in their life, such as OpenAI’s ChatGPT or Anthropic’s Claude, and is aware of the remarkable capabilities that these models offer for day-to-day tasks.

There is vast, untapped potential for LLM-based applications in network management. In particular, the long-term goal of “Intent-Based Networking” [48] is to allow network operators to state their desired operational goals for the underlying software to understand those goals, and autonomously perform the necessary end-to-end network optimization steps necessary for achieving them. The article presented in this Chapter makes a step towards achieving this long-term goal by exploring the capabilities of open-source LLMs for translating natural-language instructions to the appropriate set of network configurations.

Our reference scenario is the in-house optical network testbed provided to us by SM-Optics and operated via a REST-based controller [49, 50]. Our considered ML task is to “translate” natural-language instructions to the appropriate data structure (in JSON format) to be transmitted via the REST API. This problem, while deceptively simple, cannot be solved via naive prompting. Indeed, we show in the paper that naive prompting achieves error rates above 90%. In the paper, we address two main challenges: a) how to effectively “teach” an LLM how to operate a customized testbed, and b) how to ensure that the JSON data structures produced by the LLM conform to the API contracts of the REST controller. To effectively solve this challenge, we propose two fundamental methodological contributions: 1) we decompose the “intent translation” task into two simpler steps (Planning and Execution), and 2) we leverage LLM constraining techniques to provide syntactical guarantees on the LLM’s output. Thanks to our system design, we can reduce the error rates below 20% on our dataset. Given the extremely fast pace of advancements in LLM research, our results are likely to drastically improve in the presence of more capable models than the ones considered in the paper. Our implementation and evaluation dataset are publicly available.⁶

Differently from prior exploratory work in this field, we exclusively leverage publicly-available LLMs instead of proprietary served solutions. We believe this makes our results particularly attractive to industry applications, where data privacy is of paramount importance. Indeed, our technical contributions served as a cornerstone for a follow-up post-deadline paper, in collaboration with Huawei partners, showcasing the first experi-

⁶<https://github.com/nicoladicicco/llm-orchestrator>

mental demonstration of full lifecycle automation in optical networks using LLMs [51]. We expect to see increasing investments from academia and industry on LLM-based network automation in the imminent future.

Chapter 8: Conclusion and Future Development. This final Chapter summarizes our main technical contributions and the lessons learned from this Thesis work. We conclude by briefly describing future promising research directions and follow-up projects.

1.3. Other Contributions

My PhD work has included several other publications. As either they cover other subjects, or I was not the lead contributor, they do not form a part of this manuscript. These contributions mainly concerned optical-network optimization and collaborations with other research groups on topics adjacent to this Thesis. As a co-author, I contributed to supervising the technical contributions related to mathematical optimization and aiding in designing ad-hoc ML techniques for telecommunications problems. The list of papers I co-authored during my PhD, beyond the ones presented in this manuscript, is as follows:

- Nicola Di Cicco, Memedhe Ibrahim, Cristina Rottondi, Massimo Tornatore. Calibrated Probabilistic QoT Regression for Unestablished Lightpaths in Optical Networks, *5th International Balkan Conference on Communications and Networking (BalkanCom)*, 2022 [52]
- Nicola Di Cicco, Federico Tonini, Valentina Cacchiani, Carla Raffaelli. Optimization over time of reliable 5G-RAN with network function migrations, *Computer Networks*, 2022 [53]
- Omran Ayoub, Nicola Di Cicco, Fatima Ezzedine, Federica Bruschetta, Roberto Rubino, Massimo Nardecchia, Michele Milano, Francesco Musumeci, Claudio Passera, Massimo Tornatore. Explainable Artificial Intelligence in communication networks: A use case for failure identification in microwave networks, *Computer Networks*, 2022 [54]
- Nicola Di Cicco, Simone Del Prete, Silvi Kodra, Marina Barbiroli, Franco Fuschini, Enrico Maria Vitucci, Vittorio Degli Esposti, Massimo Tornatore. Machine Learning-Based Line-Of-Sight Prediction in Urban Manhattan-Like Environments, *17th European Conference on Antennas and Propagation (EuCAP)*, 2023 [55]
- Nicola Di Cicco, Jacopo Talpini, Memedhe Ibrahim, Marco Savi, Massimo Tornatore. Uncertainty-Aware QoT Forecasting in Optical Networks with Bayesian

Recurrent Neural Networks, *IEEE International Conference on Communications (ICC)*, 2023 [56]

- Giovanni Simone Sticca, Memedhe Ibrahim, Francesco Musumeci, Nicola Di Cicco, Andrea Castoldi, Rosanna Pastorelli, Massimo Tornatore. Selective hybrid EDFA / Raman amplifier placement to mitigate lightpath degradation in (C + L) networks, *Journal of Optical Communications and Networking*, 2023 [57]
- Faruk Pasic, Nicola Di Cicco, Marco Skocaj, Massimo Tornatore, Stefan Schwarz, Cristoph Mecklenbräuker, Vittorio Degli Esposti. Multi-Band Measurements for Deep Learning-Based Dynamic Channel Prediction and Simulation, *IEEE Communications Magazine*, 2023 [58]
- Marco Skocaj, Nicola Di Cicco, Tommaso Zugno, Mate Boban, Jiri Blumenstein, Ales Prokes, Tomas Mikulasek, Josef Vychodil, Konstantin Mikhaylov, Massimo Tornatore, Vittorio Degli Esposti. Vehicle-to-Everything (V2X) Datasets for Machine Learning-Based Predictive Quality of Service, *IEEE Communications Magazine*, 2023 [59]
- Aryanaz Attarpour, Memedhe Ibrahim, Nicola Di Cicco, Francesco Musumeci, Andrea Castoldi, Mario Ragni, Massimo Tornatore. Minimizing the cost of hierarchical optical transport network traffic grooming boards in metro networks, *Journal of Optical Communications and Networking*, 2023 [60]
- Giovanni Simone Sticca, Memedhe Ibrahim, Nicola Di Cicco, Francesco Musumeci, Massimo Tornatore. Throughput maximization in (C+L+S) networks with incremental deployment of HFAs and 3Rs, *49th European Conference on Optical Communications (ECOC)*, 2023 [61]
- José Santos, Mattia Zaccarini, Filippo Poltronieri, Mauro Tortonesi, Cesare Stefanelli, Nicola Di Cicco, Filip De Turck. Efficient Microservice Deployment in Kubernetes Multi-Clusters through Reinforcement Learning, *IEEE Network Operations and Management Symposium (NOMS)*, 2024 [62]
- Giovanni Simone Sticca, Memedhe Ibrahim, Nicola Di Cicco, Francesco Musumeci, Massimo Tornatore. Hollow-Core-Fiber Placement in Latency-Constrained Metro Networks with edgeDCs, *Optical Fiber Communications Conference and Exhibition (OFC)*, 2024 [63]
- Aryanaz Attarpour, Memedhe Ibrahim, Nicola Di Cicco, Francesco Musumeci, Andrea Castoldi, Mario Ragni, and Massimo Tornatore. Joint QoT-Aware Optimization of OTN and WDM Layers for Low-Cost Optical Metro Networks, *IEEE Inter-*

national Conference on Communications (ICC), 2024 [64]

- Gaetano Francesco Pittalà, Cristian Zilli, Nicola Di Cicco, Gianluca Davoli, Alessio Sacco. Recovering Missing Monitoring Data to Enhance Service Provisioning in the Edge-to-Cloud Continuum, *IEEE 10th International Conference on Network Softwarization (NetSoft)*, 2024
- Klevis Duka, Nicola Di Cicco, Enrico Maria Vitucci. A Study on the Use of Convolutional Networks for RF Coverage Evaluations in Urban Environments, *IEEE International Symposium on Antennas and Propagation and INC/USNC-URSI Radio Science Meeting (AP-S/INC-USNC-URSI)*, 2024 [65]
- Giovanni Simone Sticca, Memedhe Ibrahim, Nicola Di Cicco, Francesco Musumeci Massimo Tornatore. On High-Power Optical Amplification in Hollow Core Fibers for Energy Efficiency and Network Throughput Maximization, *50th European Conference on Optical Communications (ECOC)*, 2024 [66]
- Chenyu Sun, Xin Yang, Nicola Di Cicco, Reda Ayassi, Venkata Virtajut Garbhapu, Photios A. Stavrou, Massimo Tornatore, Gabriel Charlet, Yvan Pointurier. First Demonstration of Fine-Tuned LLM for Digital Twin Optical Networks: AI-Agent for Lifecycle Management and Automation, *50th European Conference on Optical Communications (ECOC)*, 2024, **post-deadline paper** [51]
- Nicola Di Cicco, Filippo Poltronieri, José Santos, Mattia Zaccarini, Mauro Tortonesi, Cesare Stefanelli, Filip De Turck. Multi-Objective Scheduling and Resource Allocation of Kubernetes Replicas Across the Compute Continuum, *20th International Conference on Network and Service Management (CNSM)*, 2024 [67]
- Oleg Karandin, Aleix Lahoz, Nicola Di Cicco, Francesco Musumeci, Massimo Tornatore. Resource-Efficient Implementation of Multiple Concurrent Tree-Based Models in P4 Switches using Feature Sharing, *20th International Conference on Network and Service Management (CNSM)*, 2024 [68]
- Qiaolun Zhang, Nicola Di Cicco, Memedhe Ibrahim, Raul Almeida Júnior, Alberto Gatto, Raouf Boutaba, Massimo Tornatore. Link Configuration for Fidelity-Constrained Entanglement Routing in Quantum Networks, *IEEE International Conference on Computer Communications (INFOCOM)*, 2025 [69]
- Giovanni Simone Sticca, Memedhe Ibrahim, Nicola Di Cicco, Francesco Musumeci, Massimo Tornatore. Hollow Core Fiber as a Long-Term Solution for Capacity Scaling in Optical Networks, *Optical Fiber Communication Conference and Exhibition (OFC)*, 2025 [70]

- José Santos, Mattia Zaccarini, Filippo Poltronieri, Mauro Tortonesi, Cesare Stefanelli, Nicola Di Cicco, Filip De Turck. HephaestusForge: Optimal Microservice Deployment across the Compute Continuum via Reinforcement Learning, *Future Generation Computer Systems*, 2025 [71]

2 | State of the Art

In this Chapter, we extensively review recent advances in ML for communication networks, with emphasis on the specific applications investigated in this Thesis. Specifically, we survey macro-areas related to the content of each main Chapter: ML for Network Optimization (Section 2.1 and Chapter 3), ML for failure identification (Section 2.2 and Chapter 4), ML for service orchestration (Section 2.3 and Chapter 5), In-Network ML (Section 2.4 and Chapter 6), and Language Models for network management (Section 2.5 and Chapter 7). For each macro-area, we compare and contrast with the papers adjacent to this Thesis’ work, highlighting the advancements brought to the state of the art.

2.1. Machine Learning for Network Optimization

Reinforcement Learning is a ML paradigm for solving sequential decision-making problems [72]. In RL, an agent interacts repeatedly with a (possibly stochastic) environment, receiving feedback in the form of scalar rewards. The optimization objective then becomes maximizing the accumulation of rewards in the long term. Thanks to its generality and flexibility, RL has become a popular computational method for learning to solve combinatorial optimization problems [73–78], with broad applications in networking [4, 15, 34, 35, 79–85].

One prominent line of investigation addressed in this Thesis concerns leveraging RL for learning specialized heuristics (or subroutines within larger heuristics) of offline routing and resource allocation problems [4, 34, 35, 79, 82, 84]. The fundamental design principle of these works is to partly or fully delegate to a RL agent the decisions on which routing paths to select, and how resources over these paths should be distributed among different traffic demands.

The seminal work in [79] was among the first to propose the possibility of using RL for solving routing problems. More specifically, this paper considers a “multicommodity-flow-like” family of routing problems, i.e., where traffic demands are treated as splittable, continuous commodities. In this context, the decisions become how to split the traffic

demands over different edges in a capacitated network for optimizing a specific objective, e.g., minimize the maximum link load or maximize the network throughput. The authors consider setting the path-flow split ratios as the actions of a RL agent, and the resulting optimization objective as a reward function. In contrast with conventional approaches based on Supervised Learning (i.e., learning to imitate optimal solutions), in RL an agent can learn to directly optimize the final objective without the need for ground-truth optimal solutions. The methodology pioneered in this work was the basis for extensive future research in Software-Defined Networking (SDN) routing, e.g., [4, 86–90].

In this Thesis, differently from the aforementioned line of research, we focus on *combinatorial* network optimization, i.e., on problems where the decisions to be made are inherently discrete. Such problems are ubiquitous in networking. Prominent examples include Routing and Spectrum assignment in optical networks [91] (where we decide how to route over a set of discrete wavelengths), Virtual Network Embedding problems [92] (where we decide how to map a logical graph on top of a physical topology), and network service orchestration [41] (where we decide which servers should process incoming traffic demands). In contrast with multicommodity-flow-like problems, combinatorial network optimization problems are inherently hard to solve (both theoretically and computationally), with problem instances of practical size being outside the capabilities of modern commercial solvers. For the above reasons, it is of great practical interest to investigate the potential of RL for learning more efficient solving algorithms.

The first contribution we will present on this topic investigates the potential of Deep Reinforcement Learning (DRL) for solving RWA problems. At the time of publication, RL has been already successfully applied for optical-network optimization, e.g., [80, 81, 93–95]. However, even though it was well-established that RL-based methods could outperform standard RWA heuristics, it was not clear how they would perform against state-of-the-art solving methods, such as the Genetic Algorithm employed in [96] and subsequent works. In contrast to prior literature, our results paint a more nuanced picture on the capabilities of RL for optical-network optimization. Our main finding is that purely RL-based algorithms achieve a middle-ground trade-off between conventional heuristics and advanced metaheuristics. Specifically, RL falls short in terms of performance, but it is generally much faster in terms of execution time. While this trade-off might sound compelling in selected scenarios, outperforming consolidated methods is a *sine qua non* for practical applicability of RL-based methods.

The above challenges prompted us to change approach, and focus our attention on how to get the best of both conventional optimization and RL. Generally speaking, stochastic Local Search algorithms and their variations are the current state-of-the-art for many

combinatorial optimization problems. For this reason, the second contribution of this Thesis on ML for optimization investigates how to improve Local Search with RL, and how does this compare with advanced metaheuristics.

Remarkably, the general idea of leveraging RL for improving stochastic Local Search algorithm dates back to more than two decades ago [97–101]. Literature extensively investigated RL for *reactive algorithms*, i.e., metaheuristics able to dynamically self-improve their performance at runtime. Thanks to several breakthroughs in modern RL computational methods, especially in DRL [102–105], the general ideas put forward in the aforementioned seminal papers have seen a great revival.

Modern representative contributions in using RL for improving stochastic Local Search algorithms are [106–109]. In [106], a DRL agent is used to learn repair heuristics for vehicle routing problems, i.e., specialized procedures for turning unfeasible solutions into a feasible one. In [107], a DRL agent is embedded into a Variable Neighborhood Search algorithm for choosing in which order should the neighbors be explored, in contrast with conventional algorithms adopting a fixed order. Similarly, in [108], a DRL agent is used to augment a Simulated Annealing (SA) algorithm. Specifically, the DRL agent implements a Local Search algorithm for refining the SA solution at each optimization iteration. Finally, in [109], a DRL agent is used to augment a SA algorithm, in a way that at each iteration, the DRL agent decides how to perturb the current solution.

In DeepLS, one of the contributions presented in this Thesis, we leverage DRL to augment a simple Local Search algorithm, with the goal of learning to select which is the best neighbor to explore. Our work brings several improvements compared to past literature in this field. Specifically, in contrast to [107, 108], which aim at learning neighborhood structures, our contribution assumes that the neighborhood is known, and the fundamental decision is how to properly select neighbors. These two approaches are complementary to each other. The work in [109] shares the most similarities with our work, also using DRL for learning to iteratively perturb solutions. Our fundamental insight is that to learn effective algorithms, we do not need to resort to a full-fledged SA algorithm, but a simple Local Search is sufficient. The reason is that SA algorithms stochastically decide whether to accept or reject non-improving moves, depending on a “temperature” parameter that is slowly annealed with the number of iterations [110]. We find this stochastic aspect detrimental for DRL learning since, from the point of view of the agent, the randomized behavior of SA is part of the (unobservable) stochasticity of the environment. Moreover, SA requires a notoriously large number of iterations to converge. In routing problems, evaluating the objective function requires provisioning the complete traffic matrix, which can become prohibitively expensive in the long run. For these reasons, we find it simpler

and more effective to let the agent autonomously decide whether to explore potentially non-improving solutions, thus learning autonomously how to balance intensification and diversification during the Local Search process.

In terms of the specific networking use-case, the works that share the most similarity to DeepLS is our previous contribution to DRL for RWA, and the methodologies proposed in [82, 84]. More specifically, in [82, 84], the authors proposed leveraging Graph Neural Networks (GNNs) to learn heuristic algorithms for combinatorial traffic engineering problems, outperforming state-of-the-art solving methods such as DEFO [111, 112]. The main driving factor behind using GNNs is their capability to perform inference on network topologies of arbitrary size, enabling generalization of the proposed algorithms to network topologies different from training. In DeepLS, we adopt a significantly simpler approach, adopting a Deep Sets-like [43] lightweight neural network architecture that still enables generalization to arbitrary-sized networks, but does not embed the topology information like GNNs. This not only results in shorter training and inference times but also, surprisingly, in improved performance. We posit that the reason for this counterintuitive result is that it is difficult for a GNN to abstract information about paths and flows from a given input network topology, resulting in suboptimal learning. When applied to RWA problems, DeepLS outperforms our heavily-specialized Genetic Algorithm, whose performance was out of the reach of past RL-based algorithms, including our own prior contributions. We hope that the general insights we gained from developing DeepLS will be useful to the community at large.

2.2. Machine Learning for Failure Identification

Fault detection and identification are among the most prominent use cases of ML for network management. In the whole management pipeline, prompt and accurate identification of failures allows network operators to respond quickly with the appropriate countermeasures. For example, failures caused by software misconfigurations may be solved remotely, while hardware failures necessitate physical interventions from technicians.

The contributions presented in this Thesis on the topic focus on the general problem of classifying hardware failures in microwave radio equipment, in the context of a collaborative project with SIAE Microelettronica [113]. The first investigations on applying ML to this problem date back to the 90s, e.g., [114], and have been popularized in recent years making use of modern ML modeling solutions and tooling [1, 54, 115–120].

Our contributions take a significant departure compared to the state of the art. Specifically, prior work extensively investigated how to train effective ML models for fault

classification given a training set [115–117, 119, 120]. These contributions have been fundamental to the field, and have established ML models as the de facto solution for fast and precise fault identification. Though fundamental, there are two main fundamental issues not yet addressed. First, we argue that the majority of the costs sustained by a network operator for deploying a ML solution are not from model training, but from data collection and annotation. In contrast to popular ML domains such as Computer Vision or Natural Language Processing, curated data for fault identification is not only scarce, but also expensive to annotate, as it requires domain experts with deep knowledge of the networking equipment. Second, ML-based fault identifiers act fundamentally as black-box hard classifiers, with no guarantees on their performance once deployed. Even though we can measure the average performance of a ML model on a test set, we do not have any principled measure of information about how reliable future *individual predictions* will be.

To solve the problem related to the cost of collecting and annotated data, we proposed shifting the paradigm of ML-based fault identification from model-centric to data-centric, i.e., by focusing on methods revolving around how to curate and improve as much as possible the training data. This approach is rooted in the fact that ML models are intrinsically *garbage in \rightarrow garbage out*, i.e., a ML model can be as most as good as its training data. We adopt two methodologies for data-centric ML: using generative models to augment an existing training dataset, and leveraging Active Learning for guiding the data collection and annotation procedure after the ML model is deployed. We now briefly review relevant literature on both of these topics.

For data augmentation with generative models, the two main solutions investigated in the literature are Generative Adversarial Networks (GANs) [121–129] and Variational Autoencoders (VAEs) [130–134]. Both these methodologies aim to fit complex statistical distributions to a representative training set, and then sample from them. Although GANs have proven successful in synthetic image generation [121], there is presently no formal theoretical model on why GANs should outperform VAEs for a specific synthetic data generation task. One fundamental practical difference in GANs and VAEs is in their training algorithm: while GANs require finding an equilibrium between two competing neural networks, VAEs can be trained with conventional stochastic gradient descent, and it is therefore easier to achieve convergence.

The common problem considered by data augmentation paper relates to increasing the number of samples for the poorly-represented class in the training dataset. This is because ML models tend to overly favor the most represented classes in the training set, leading to poor performance on the least-represented ones [135]. Our work improves over prior contributions in two main aspects. First, we leverage *conditional* generative models [136]

for fine-grained control of which failure classes to generate at runtime. Generative models suffer from the same problem as conventional ML model, i.e., they tend to favor over-represented classes. Because of this, sampling from the training data distribution can result in low-quality synthetic samples for poorly-represented classes. Instead, conditional generative models explicitly inject the class information during the training and inference stage, allowing the generative model to disentangle the specific features for each class. In practice, for our application, this has translated into improved performance across a broad set of state-of-the-art baselines. Second, while prior literature focuses on low-dimensional tabular data, our work is a successful application of synthetic data generation on a real-world hardware failure dataset consisting of more than 160 features.

Active Learning refers to data labeling methodologies that leverage the predictions of a ML model to make informed decisions on which data point should be annotated. The final objective is to train a top-performing ML model with the least amount of data possible. In the broader context of network management, only a few works have put a strong emphasis the potential of Active Learning (AL) [117, 137–139]. Specifically, [138] leverages Gaussian Processes to generate data close to the decision boundary of the model. In [117], the authors leverage Unsupervised Learning to create pseudo-labels for anomalous data. In [137] the authors propose a stream AL approach that, given a new unlabeled data point, combines the ML model’s uncertainty with the distance from the new data point to the current training data set for deciding whether to label it or not.

In our contribution, we started from the fundamental observation that an AL algorithm at deployment time should not come at the cost of using a suboptimal ML model architecture. Since tree ensembles are known to be the best-performing algorithms on tabular data [140], differently from prior literature, we aimed at developing an AL-based annotation targeted specifically for tree models. For this reason, we leveraged uncertainty decomposition in tree ensembles [141] to estimate which samples, from a pool of unlabeled failures, would contribute more to the knowledge increase of the model. Moreover, motivated by the fact that multiple human annotators are available at the same time, we developed an ad-hoc Batch AL for concurrent labeling of multiple annotated samples, with the goal of maximizing the informativeness of new data while avoiding redundant samples as much as possible. Our algorithms have been successfully deployed in the field, and have resulted in performance gains aligned with the ones reported in the paper.

Our contributions to reliable fault classification lie in the broader topic of Explainable Artificial Intelligence (XAI). The high-level goal of XAI is to make black-box models naturally interpretable and trustable by humans. Uncertainty quantification is a fundamental component of XAI, as it allows a human user to quantitatively estimate the risk

associated to accepting a prediction.

Recent effort in XAI applied to network management, including fault classification in microwave networks, have focused on deriving *feature attributions* of trained models [54, 118, 142, 143]. In particular, these algorithms at estimating how the response of a trained ML model is influenced by a change in the input feature values, the most popular approaches being LIME [144] and SHAP [145].

Our contributions are orthogonal to prior work. Instead of attempting to explain the complex decision function learned by a ML model, we instead focus on guaranteeing precise statistical properties on their outputs. Specifically, we are interested in the problem of training *probabilistic* fault classifiers, i.e., ML models returning a statistical measure of confidence in their predictions. Critically, we want this measure of confidence to be *calibrated*, i.e., to be a real probability. Indeed, though many ML models output normalized scores, they in general do not map to proper probabilities, often resulting in underestimating the true uncertainty associated to a prediction. In the context of fault management, this particular issue is of critical importance, since the user may be led to believe a fault prediction to be absolutely certain, while in reality the predictive confidence should have been significantly lower.

In our contribution, we solve the aforementioned problem by leveraging recent advances in the field of Conformal Prediction (CP), which, at the date of publication, was yet to be applied in fault classification for microwave networks. The fundamental idea of CP is to perform a post-hoc “calibration” of a trained ML model on a held-out dataset, such that a heuristic measure of uncertainty produced by the ML model is reshaped to match the empirical distribution of the data. Specifically, in our paper, we leverage Venn-Abers predictors, an algorithm for converting any black-box ML model into a probabilistic classifier with formal theoretical guarantees. We apply this methodology in the context of As-Soon-As-Possible (ASAP) prediction: given a stream of telemetry data from microwave equipment, we want to return a fault prediction as soon as the probability of correct classification exceeds a user-provided safety threshold. We therefore propose a new, practical angle for reliable deployment of ML-based fault detectors, and we empirically demonstrate its effectiveness on a real-world dataset.

In summary, the contributions of this Thesis build upon prior literature by investigating the challenges arising before and after training a ML model, and proposing novel methodologies for solving them. The effectiveness of our solutions has been demonstrated both on real-world datasets and in field operations. We hope that the problem statements we distilled, and the data we published, will be of inspiration for future research.

2.3. Machine Learning for Service Orchestration

The integration of Cloud and Edge resources plays a fundamental role in modern communications networks [41]. In particular, the heterogeneous QoS requirements of different services (e.g., in terms of network throughput, computational resources, latency, etc.) require an intelligent *network orchestration* layer for properly assigning services to the most appropriate processing nodes.

This general problem of computation offloading is well-studied in the literature, including solving methodologies based on ML. Similar to Section 2.1, we consider leveraging ML for learning optimization algorithms. The fundamental difference is that here we are dealing with an *online* optimization problem, where decisions are taken sequentially as service requests dynamically arrive and depart. The necessity for near-real-time decision-making makes conventional offline optimization approaches (e.g., ILP, metaheuristics) inapplicable, opting instead for fast rule-based heuristics. In this context, ML-based approaches, especially RL, have received significant attention as a means for outperforming conventional rule-based heuristics without compromising on decision-making speed.

The literature has extensively investigated both centralized [146–159] and decentralized [160–165] ML-based approaches for service orchestration and resource allocation. The general task investigated in these works can be summarized as follows: given the network’s state and the current incoming demands, we must decide over which computational node to offload a service such that a utility function is maximized in the long term. Common utility functions include the service blocking probability, QoS metrics such as throughput or latency, and the network’s energy consumption. In the case of a centralized approach, it is assumed that a centralized orchestrator entity has a global view of the network’s state, while in the decentralized approach, individual entities (e.g., groups of nodes) must reach a consensus on how to distribute the services, possibly jointly optimizing non-cooperative goals. The contributions presented in this Thesis fall in the latter assumptions, i.e., centralized orchestration.

Though the state of the art extensively demonstrates the potential of RL-based approaches for outperforming conventional heuristics, we identify and propose solutions to two main fundamental shortcomings of current literature.

First, the proposed approaches cannot trivially be applied to networks different from training without either re-training from scratch or fine-tuning. This is because RL-based methodologies heavily rely on neural-network architectures with fixed input-output dimensionalities. The work in [161] raised this fundamental issue and proposed a countermea-

sure, which, however, still requires re-training. In the context of Edge-Cloud computing, it is expected that the set of available computational resources will stochastically evolve with time (e.g., due to random connections / disconnections of edge devices); it is therefore of paramount importance for ML-based method to immediately adapt to new network conditions, ideally without the need for retraining. We solve this issues by putting forward RL methods based on Equivariant Neural Networks, namely Deep Sets [43]. In a nutshell, these models assume that the inputs and outputs satisfy certain symmetrical properties, and this allows the model to support arbitrarily sized inputs and outputs without the need for retraining or changing the architecture. Since the inference complexity of these models scales linearly with respect to the network size, they make a promising candidate for implementing practical and generalizable RL-based policies. We empirically demonstrate that our RL-based orchestrator compares favorably against standard rule-based heuristics, generalizing its learned policy to networks several times larger than training.

Second, the current literature abstracts service orchestration as a single-objective optimization problem when, in practice, multiple conflicting objectives (e.g., blocking probability, latency, energy consumption, etc.) are at stake. The standard approach is to merge multiple objectives into single ones via, e.g., linear combinations, but this approach fails to capture the trade-offs arising from the (often non-linear) interactions between competing objectives. Our contributions address this issue by putting forward proper Multi-Objective Reinforcement Learning (MORL) with the goal of deriving a set of Pareto-optimal policies, i.e., RL policies achieving a specific trade-off between the objectives. Though this problem has been considered in an offline planning context, it was not yet properly addressed in online optimization, e.g., [166–168], where decisions must be taking sequentially. To solve this problem, we extend our previous contributions with a simple and efficient MORL algorithm, and we build a realistic model, based on profiling of real hardware devices, of node energy consumption as a function of CPU load. This allows us to explore the different trade-off between service acceptance probability and network energy consumption and derive a set of RL policies that perform well over a broad set of possible preferences, as opposed to rule-based heuristics targeting one specific trade-off.

In summary, the problem statements and main challenges in RL for service orchestration greatly overlap with the ones in RL for offline optimization. Differently from offline problems, online optimization requires dealing with the dynamics of the network’s state (e.g., stochastic service arrivals and departures), increasing the complexity of the problem. In this context, there is a rising need for ML models that are generalizable and adaptable, especially in the context of multi-objective optimization problems. We hope that our contributions will help guide future investigations in this field.

2.4. In-Network Machine Learning

Programmable Network Interface Cards (NICs) and network switches have received significant attention in recent years. There are several important reasons behind their great success. Offloading customized network functions (e.g., traffic classification, load balancing) to specialized hardware opens up an incredible amount of opportunities for network acceleration. Moreover, programmability enables post-deployment reconfiguration of the traffic processing logic, without being tied to specific vendor’s hardware or non-configurable software environments.

In this context, there have been a great interest in offloading inference of trained ML models to packet-processing hardware. This line of investigation is motivated by the flexibility of simple ML models (e.g., small neural networks or decision trees) for learning complex decision rules, while also benefitting from line-rate packet processing speeds [7, 8, 169–174]. More specifically, the contemporary literature extensively investigated how to efficiently map commonly-used ML model into programmable switches (most prominently, P4 switches [175]) subject to the architectural constraints of packet-processing hardware, such as the absence of floating-point computation and restricted control-flow loops [8, 172, 173]. At the time of writing, specialized strategies for mapping models such as feedforward neural networks [171], decision trees, and random forest [8, 173], as well as automated ML model architecture search algorithms for automated deployment [172].

The contributions presented in this thesis are orthogonal to prior research. Specifically, while prior work investigates how to efficiently train and deploy models on programmable switches, we investigate how to efficiently re-train and adapt train models on new data. Indeed, it is well-known that the performance of ML models is susceptible to *concept drift*, i.e., to changes in the training distribution of the test data with respect to the training data [176, 177]. Networking is no exception to this rule: indeed, as we empirically demonstrate in the presented contribution, traffic patterns are bound to change over time, and this will have a negative impact on the downstream performance of trained ML models. Naive online updates of the ML models are bound to cause *catastrophic forgetting*, i.e., the ML model will forget past accumulated knowledge to fit new data. This fundamental issue prevents deploying the ML model in real-world application scenarios.

To solve these fundamental issues, we propose leveraging methodologies from Continual Learning for lifelong updating on a trained in-network ML model and from Active Learning to reduce as much as possible the cost of labeling new data. Our preliminary results on incremental traffic classification show good capabilities of adaptation (a few % less accuracy than an “oracle” model), while requiring only less than 2% of the training data.

We leave ample room for future research, including more advanced Continual Learning and Active Learning methodologies, and experimental evaluation on real P4 switches.

2.5. Language Models for Network Automation

LLMs achieved unprecedented breakthroughs in natural language processing. At the time of writing, the most-capable models such as OpenAI’s GPT-4 [47], Google’s Gemini [178], Anthropic’s Claude [179], or Meta’s Llama 3 [180], are achieving near-human performance in a variety of tasks (e.g., question answering, math [181], coding [182]) and their capabilities are expected to significantly improve in the near future. For these reasons, exploring the capabilities (and limitations) of LLMs for network management, especially in the context of intent-based networking, is currently a hot topic of research. Though the topic is still in its infancy, several major contributions have already been made [183–187].

Contemporary literature is currently investigating the following two major research challenges in LLMs for network automation: using LLM to produce natural-language explanations of the network’s state [183–186], and using LLM to implement a natural-language interface between the end-user and the network [183, 184, 186, 187]. The paper presented in this Thesis makes a significant system design contribution to the second problem.

Literature using LLMs to interface with network equipment [183, 184, 186, 187] aims to solve this problem via “prompt engineering” techniques, i.e., by crafting specialized natural-language instructions steering a pre-trained LLM to the desired outputs, e.g., code snippets, network configurations, and so on. This work is motivated by the need to automate repetitive network management tasks, such as configuration templating, data analysis and boilerplate coding, as well as reducing the barrier of entry for interfacing with a complex software-defined network. Preliminary work in this field investigated the potential of state-of-the-art closed-source models, like GPT-4, for solving these tasks. Our preliminary contribution to this field improve over recent work in several aspects. Focusing on a use-case on optical-network configuration, we make a case about using open-source and self-hosted LLM, since control of their data is of paramount importance for network operators, and it has happened several times that closed-source LLM have been subject to confidentiality data breach, e.g., [188]. Moreover, based on the observation that a substantial portion of errors of LLM are *syntactical errors* (e.g., when synthesizing the payload of an API call), we leverage formal grammars to constrain the LLM’s output at runtime, guaranteeing that the output will be consistent with a formal language. Then, we adopt a task decomposition approach, such that complex instructions are first split into multiple sub-tasks (each corresponding to one API call), which are then solved one

at a time. Our contributions can reduce the error rate of a medium-size open-source LLM from over 90% to under 20%. Finally, in contrast to prior investigations in LLM-based optical-network management, our evaluation dataset, based on a real-world SDN API, is publicly available.

All in all, the main argument discussed in our contributions is that LLMs are not plug-and-play tools, and non-trivial ad-hoc methodologies are required for them to work well in the context of network management. More than this, the field is in dire need of standardized benchmarks and performance indicators for evaluating LLM-based network management systems. We hope that our contributions will help to shape this rising field.

3 | Reinforcement Learning for Network Optimization

This Chapter presents several ML-based algorithms for solving classical combinatorial network optimization problems. We focus on RL, since it is currently being investigated as a competitive alternative to conventional solving methods, e.g., metaheuristics and commercial solvers. Specifically, in Section 3.1, we benchmark end-to-end RL against a customized metaheuristic for the RWA problem, highlighting advantages and shortcomings of ML-based solutions over conventional algorithms. Following up this line of investigation, in Section 3.2 we propose integrating RL with a simple Local Search algorithm. Our numerical experiments suggest that augmenting simple general algorithms such as Local Search with lightweight intelligence allows us to attain comparable or better performance than state-of-the-art, handcrafted algorithms.

3.1. On Reinforcement Learning for Routing and Wavelength Assignment

This paper considers the problem of learning end-to-end constructive heuristics for Routing and Wavelength Assignment problems with Deep Reinforcement Learning. At the time of publication, DRL has already been explored as a promising technology for solving routing and resource allocation problems in communications networks, e.g., [80, 83, 93]. The purpose of this paper is twofold. First, we aim to provide a self-contained overview of DRL as a methodology. Second, we aim to numerically assess how purely DRL-based solving algorithms for RWA compare against well-established solving methods, namely, a heavily customized Genetic Algorithm that has been applied with great success to a variety of optimization problems in our research group [96]. This is in contrast with prior work on the literature, which compared DRL-based approaches only against simple heuristics. Our numerical results highlight an interesting trade-off that, before this work, has been overlooked in the literature. The trade-off is as follows: though DRL-based algorithms provide shorter execution times compared to metaheuristics, they fall short in

terms of solution quality. In other words, customized metaheuristics are still the preferred solving method if computational times are not a strict requirement. Our results highlight that DRL alone is likely to be insufficient for developing solving methods with tangible advantages compared to classical approaches from Operations Research. This empirical conclusion will be the basis for the next paper presented in this Chapter.

On Deep Reinforcement Learning for Static Routing and Wavelength Assignment

Nicola Di Cicco, Emre Furkan Mercan, Oleg Karandin, Omran Ayoub,
Sebastian Troia, Francesco Musumeci, and Massimo Tornatore

(Invited Paper)

Abstract—Deep Reinforcement Learning (DRL) is rising as a promising tool for solving optimization problems in optical networks. Though studies employing DRL for solving static optimization problems in optical networks are appearing, assessing strengths and weaknesses of DRL with respect to state-of-the-art solution methods is still an open research question. In this work, we focus on Routing and Wavelength Assignment (RWA), a well-studied problem for which fast and scalable algorithms leading to better optimality gaps are always sought for. We develop two different DRL-based methods to assess the impact of different design choices on DRL performance. In addition, we propose a Multi-Start approach that can improve the average DRL performance, and we engineer a shaped reward that allows efficient learning in networks with high link capacities. With Multi-Start, DRL gets competitive results with respect to a state-of-the-art Genetic Algorithm with significant savings in computational times. Moreover, we assess the generalization capabilities of DRL to traffic matrices unseen during training, in terms of total connection requests and traffic distribution, showing that DRL can generalize on small to moderate deviations with respect to the training traffic matrices. Finally, we assess DRL scalability with respect to topology size and link capacity.

Index Terms—Deep Reinforcement Learning, Routing and Wavelength Assignment, Genetic Algorithm, Optimization.

I. INTRODUCTION

In recent years, several applications of Machine Learning (ML) in optical networks have been developed, such as Quality of Transmission (QoT) estimation [1], failure management [2] and traffic prediction [3], demonstrating the ability of ML models to extract useful information from the monitoring data available in optical networks.

Among the various tools offered by ML, Deep Reinforcement Learning (DRL) is attracting particular attention as a promising tool for tackling complex optimization problems for resource allocation in optical networks [4]. However, even though DRL has shown remarkable results in tasks such as playing games [5]–[7] and continuous control [8], [9], investigations on DRL for solving optimization problems in optical networks have started only in recent years [10]–[13]. What makes DRL competitive with respect to both supervised and unsupervised learning is its capability to learn directly from experience with little to no prior data, while driving optimization towards a user-specified goal. These characteristics

make DRL a particularly attractive solution for scenarios in which it is difficult, if not impossible, to acquire well-defined training data.

Nevertheless, there are many potential difficulties in getting the best performance out of DRL and in interpreting the obtained results, ranging from hyperparameter tuning to specific algorithm implementations [14]. Furthermore, by definition, a DRL-based solution method is trained on a limited subset of problem instances, such as a subset of all the possible traffic matrices in an optical network. Even though DRL can show some generalization capabilities with respect to problem instances not seen during training, assessing them a-priori is not a trivial task [15], [16]. In realistic application scenarios, if the problem instances differ too much with respect to the ones used in training, DRL may suffer from severe performance hits, to the point of requiring a new training phase starting from scratch. This introduces a significant computational burden, as training can take hours to days on modern hardware platforms.

Given these significant challenges in terms of generalization, in this study our aim is to evaluate whether DRL for optimization problems in optical networks can be a competitive alternative with respect to existing and well-established solution methods. To this end, we focus on the Routing and Wavelength Assignment (RWA) problem, a well studied problem in the optical-networking literature for which, even though there are many effective solution methods [17]–[19], more efficient algorithms are always of interest. Therefore, the RWA problem will be considered as a representative case study on which to train new DRL-based solution methods and evaluate DRL performance with respect to existing methods. In fact, we note here that comparing DRL only against greedy heuristics (as mostly done in literature) can result in an overly optimistic assessment of its actual performance. Hence, in this study, the proposed DRL methods will be compared not only to greedy heuristics, but also to a state-of-the-art metaheuristic (a Genetic Algorithm, whose performance greatly outperforms greedy heuristics), and with Integer Linear Programming (ILP), which is used to establish an optimality bound.

We also assess the sensitivity of DRL to differences between the traffic matrices seen during and after training. In fact, traffic matrices used at training time are typically drawn from a probability distribution which is assumed to be known a-priori (e.g., from traffic forecasts), but may be different in practical use-cases. We aim to evaluate whether or not it is convenient to employ DRL in place of other state-of-the-art

N. Di Cicco, E. Furkan Mercan, S. Troia, F. Musumeci and M. Tornatore are with the Department of Electronics, Information and Bioengineering (DEIB), Politecnico Di Milano, Italy. E-mail: {name}.{surname}@polimi.it

O. Ayoub is with the University of Applied Sciences of Southern Switzerland, Switzerland. E-mail: omran.ayoub@supsi.ch

solution methods, when traffic matrices differ with respect to the training ones (e.g., because of a wrong forecast) in terms of total connection requests and traffic distribution. Moreover, we assess the scalability of our proposed approach with respect to topology size and link capacity.

The remainder of this paper is organized as follows: in Section II we provide an overview of the recent works on DRL for optimization in optical network, and highlight some of the main advancements on this topic in the literature. In Section III we discuss background concepts on DRL along with the main elements of state-of-the-art DRL algorithms, with particular focus on the Proximal Policy Optimization (PPO) algorithm. In Section IV we describe the RWA problem and the considered baselines (i.e., greedy heuristics, Genetic Algorithm and ILP). In Section V we illustrate the proposed DRL approach for the RWA problem. In Section VI we benchmark the performance, in terms of blocking probability and computing times, of the proposed DRL approach against the baselines, we assess its ability to generalize to traffic matrices not seen during training and its scalability to larger problem instances, in terms of network size and link capacities. Finally, we outline conclusions and future research directions in Section VII.

II. RELATED WORK

The application of DRL algorithms in telecommunications is a fertile area of research that is giving rise to many promising results [20]. However, works applying DRL to problems in optical networks are relatively few. In the following, some of the main works regarding the application of DRL in telecommunications networks, together with their most significant achievements, are briefly summarized.

In [10], an application of DRL to routing problems in Software Defined Networking (SDN) is developed. The proposed DRL-based algorithm is able to dynamically produce routing configurations minimizing the overall delay. The proposed algorithm is an off-policy, actor-critic, deterministic policy gradient algorithm, for which an observation is defined as the traffic matrix.

In [21] authors propose a DRL-based approach for routing in optical networks. The DRL agent is tasked to route connection requests among candidate paths. A novel state representation is engineered, consisting of the states resulting from applying all possible actions to the current state. The proposed approach is able to outperform both baseline heuristics and other DRL agents using different state representations.

In [11] authors address routing in Optical Transport Networks (OTNs), with emphasis on generalization to topologies similar to the one seen at training time. Authors employ Deep Q-Learning [5], in which the DNN is implemented as a Graph Neural Network (GNN) [22]. The objective is the maximization of bandwidth utilization up to the first blocked request, meaning that long-term minimization of the blocking rate, as in our work, is not taken into account. The proposed approach outperforms vanilla DQN and greedy heuristics also for network topologies not seen during training.

In [12] authors propose a custom version of the Asynchronous Advantage Actor Critic (A3C) [23] algorithm for

dynamic Routing, Modulation and Spectrum Assignment (RMSA) in Elastic Optical Networks (EONs). After pre-computing a number of candidate paths using K-shortest-paths, the action space is defined in such a way that the agent is able to pick one among the first j available spectrum blocks in each of the candidate paths, where j is an algorithm parameter. An observation is defined such that it provides information about the current request and the spectrum utilization in each of the candidate paths. The algorithm is shown to outperform baseline greedy heuristics such as SP-FF and KSP-FF.

In [13] authors, to address the scalability issues of the algorithm developed in [12], propose a cooperative multi-agent framework for service provisioning in inter-domain EONs, in which agents are based on the Advantage Actor Critic (A2C) algorithm. Cooperation between the agents is achieved by sharing information regarding spectrum utilization between different domains, and it is shown to outperform both baseline heuristics and the results from [12].

In [24] authors develop a transfer learning framework for DRL in optical networks to mitigate the retraining needs and to provide better generalization. Authors evaluate their approach on transfer between different RMSA topologies, and from RMSA to service function chain provisioning. Their proposed approach achieves overall lower training times and performance performance similar to or better than heuristic baselines.

In [25], authors devise a Multi-Agent DRL (MA-DRL) for inter-domain RMSA, in which each agent acts as a domain broker. Agents operate independently on an abstract topology composed of border nodes and virtual nodes, the latter representing the intra-domain networks. The proposed approach manages to outperform the considered heuristic baselines.

In [26] authors develop a DRL-based admission control policy for network slicing in 5G Radio Access Networks (RANs). For each slice request, the DRL agent receives a penalty if either the request is rejected, or cannot be scaled up when needed. The proposed approach is able to outperform the considered baselines, resulting more robust to variations in the penalty values and in the service distribution.

While all of the mentioned works attain remarkable results and constitute significant advances in the applications of DRL to optical networks, each fails to fully consider at least one of the following: 1) an assessment on the generalization to unseen previous problem instances, or 2) a comparison against stronger algorithms than baseline greedy heuristics, such as state-of-the-art metaheuristics and ILP-based approaches. We argue that a thorough assessment of these two points is paramount in order not to overestimate the performance of DRL-based solution methods, and to fairly evaluate their pros and cons. In this work, we have developed DRL-based solution methods based on the state-of-the-art PPO algorithm, and we have extensively benchmarked them against both greedy heuristics and a state-of-the-art Genetic Algorithm.

III. BACKGROUND: DEEP REINFORCEMENT LEARNING

The field of Reinforcement Learning (RL) is inspired by behavioral psychology: an RL agent interacts with an environment, the latter typically represented as a Markov Decision

Process (MDP), in order to learn a policy that maximizes the expected sum of rewards over a time horizon [27].

Let \mathcal{A} be the action space and \mathcal{S} be the state space. A policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ defines the behavior of the learning agent. In particular, $\pi(a_t = a | s_t = s)$ defines the probability of selecting action $a \in \mathcal{A}$ given the state $s \in \mathcal{S}$ at time step t .

The goal of a RL agent is to maximize the expected sum of rewards, known as the return, over a time horizon T . The discounted return at time step t is defined as follows:

$$G_t = \sum_{k=t+1}^T \gamma^{k-t-1} r_k \quad (1)$$

where T is the time horizon, r_k is the reward at time step k and $\gamma \in [0, 1]$ is the discount factor. The magnitude of the discount factor determines the importance of future rewards in relation with immediate rewards.

The value function of policy π can be defined as the expected return when starting from state s and acting according to policy π , as follows:

$$V_\pi(s) = \mathbb{E}_\pi(G_t | s_t = s) \quad (2)$$

Similarly, the action-state value function, also known as Q-value, of policy π is defined as the expected return when starting from state s and taking action a , following policy π for all the following states:

$$Q_\pi(s, a) = \mathbb{E}_\pi(G_t | s_t = s, a_t = a) \quad (3)$$

The difference between the action-state value function and the state value function for a given state-action pair is defined as the advantage function:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \quad (4)$$

A policy π is said to be better than policy π' , that is, $\pi \geq \pi'$, if and only if $V_\pi(s) \geq V_{\pi'}(s)$, $\forall s \in \mathcal{S}$. All optimal policies π_* share the same value function $V_*(s)$ and action-state value function $Q_*(s, a)$, defined as follows:

$$V_*(s) = \max_{\pi} V_\pi(s) \quad (5)$$

$$Q_*(s, a) = \max_{\pi} Q_\pi(s, a) \quad (6)$$

It can be demonstrated that a policy that acts greedily with respect either to $V_*(s)$ or $Q_*(s, a)$ is an optimal policy.

Traditional RL algorithms, like Q-Learning and SARSA, store policies and value functions in tables. However, many practical environments exhibit a number of states so large that tabular methods become no longer feasible in terms of memory occupation. Hence, in DRL, policies and value functions are parameterized via function approximators, i.e., Deep Neural Networks (DNNs), leveraging the ability of DNNs to generalize to states unseen during training. Assessing and improving the generalization capabilities of DRL to unseen states is currently an active research field [15], [16].

Many state-of-the-art DRL algorithms include the following two fundamental building blocks: 1) a policy estimation phase, in which an approximation of the value function is computed, and 2) a policy improvement phase, in which the trained policy is optimized according to the value function estimation.

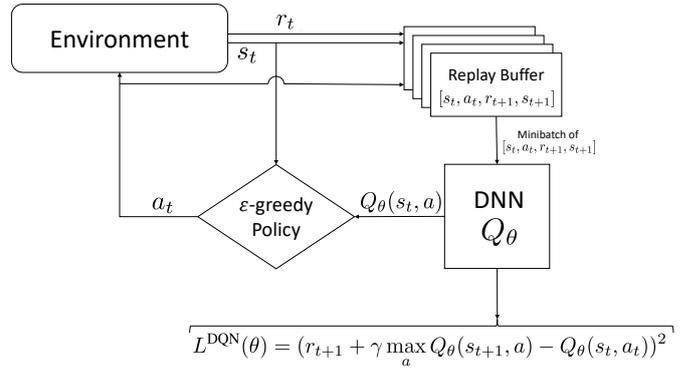


Fig. 1. Illustrative diagram of the main DQN elements, highlighting the computation of the DQN loss function.

Within such a framework, RL algorithms can be categorized as either on-policy or off-policy algorithms. In on-policy algorithms, the trained policy is used to gather samples for the policy evaluation phase. In off-policy algorithms, a behaviour (usually exploratory) policy is used to gather samples for the policy evaluation phase instead of the trained policy. Typically, off-policy algorithms are more sample-efficient than on-policy algorithms as they can reuse past experiences [5], thus they are useful in environments that are expensive to evaluate, though they tend to have slower convergence times than on-policy algorithms.

Furthermore, DRL algorithms can be further divided among policy-based, value-based and Actor-Critic. Policy-based algorithms store only a representation of the trained policy. Value-based algorithms store only a representation of the value function, from which the policy is derived. Finally, Actor-Critic algorithms store both a representation of the policy (the Actor) and a representation of the value function (the Critic).

In the following, we provide a brief overview of two relevant classes of DRL algorithms, Deep Q-Learning (DQN) and policy gradient algorithms, the latter focusing on PPO, which employed for numerical evaluation in this work.

A. Deep Q-Learning (DQN)

DQN [5] and its variants [28]–[30] are off-policy, value-based algorithms. The aim of DQN algorithms is to learn an approximation of $Q_*(s, a)$, parameterized as $Q_\theta(s, a)$ by a DNN, where θ are its parameters. The learned policy is a deterministic greedy policy, which chooses the action corresponding to the highest $Q_\theta(s, a)$ estimate. Being off-policy, a behaviour policy is used to gather the samples used for updating the Q-values estimates. An example of such a behaviour policy is an ϵ -greedy policy, which either acts greedily with respect to $Q_\theta(s, a)$ with probability $1 - \epsilon$ or randomly with probability ϵ .

The Q-Learning algorithm [31] defines the following update rule, deriving from the Bellman optimality equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (7)$$

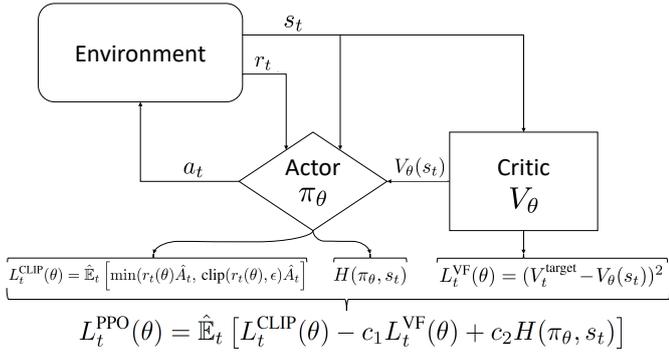


Fig. 2. Illustrative diagram of the PPO Actor-Critic architecture in which the Actor and the Critic share the same parameters θ , highlighting the computation of the individual terms appearing in the full loss function.

where α is the learning rate, which controls the magnitude of the updates.

DQN makes use of a technique known as experience replay: the behaviour policy is used to gather samples, to be stored in a replay buffer. Then, a minibatch of samples is sampled randomly from the replay buffer. For each sample, the update value of (7) is computed as the following loss function:

$$L^{\text{DQN}}(\theta) = (r_{t+1} + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t))^2 \quad (8)$$

At this point, the DNN parameters θ are updated via minibatch gradient descent. An illustrative diagram of the main DQN elements is shown in Fig. 1.

B. Proximal Policy Optimization (PPO)

Policy gradient algorithms are among the most relevant in the field of DRL. Let π_θ be a stochastic parameterized policy with parameters θ (i.e. a DNN, where the parameters are the weights of the neural network). The expected return given policy π_θ can be expressed as follows:

$$J(\theta) = \mathbb{E} \left[\sum_{k=1}^T r_k \right] \quad (9)$$

The objective of policy gradient is to optimize the parameters θ such that the expected return is maximized. Policy gradient algorithms achieve this goal by computing an estimate of the gradient of (9) and running a stochastic gradient ascent algorithm. In the literature, a plethora of different gradient estimators has been developed [32]. One of the most commonly used gradient estimators is the following:

$$\hat{g} = \hat{\mathbb{E}}_t [\nabla_\theta \log \pi_\theta(a_t | s_t) \hat{A}_t] \quad (10)$$

which, in practical implementations, results from the differentiation of the following loss function:

$$L^{\text{PG}}(\theta) = \hat{\mathbb{E}}_t [\log \pi_\theta(a_t | s_t) \hat{A}_t] \quad (11)$$

where \hat{A}_t is an estimate of the advantage function, and $\hat{\mathbb{E}}_t$ indicates an empirical average over a batch of samples. In particular, methods for computing accurate estimates of the advantage function, such as Generalized Advantage Estimation

[32], require an estimate of the value function. Therefore, computing an approximation of this expression would in principle require two DNNs: the "Actor", parameterizing the policy, and the "Critic", parameterizing the value function. In practical application scenarios a common design choice is to let the Actor and the Critic share parameters, given that they most likely would learn similar features.

In this work, an implementation of Proximal Policy Optimization (PPO) [33], one of the main state-of-the-art policy gradient algorithms, will be used for training the agents. PPO is an on-policy algorithm, since the trained policy is used to gather samples, with the following loss function:

$$L_t^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), \epsilon) \hat{A}_t \right) \right] \quad (12)$$

where $r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$ is the ratio between the policy before and after the parameters update. The clip function, which depends on the hyperparameter ϵ , prevents the occurrence of too large parameter updates that would destabilize the learning process. The complete PPO loss function at each time-step reads as follows:

$$L_t^{\text{PPO}}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 H(\pi_\theta, s_t) \right] \quad (13)$$

where $L_t^{\text{VF}}(\theta) = (V_t^{\text{target}} - V_\theta(s_t))^2$ is the squared-error loss of the value function estimator, $H(\pi_\theta, s_t)$ is the entropy of policy π_θ in state s_t , and c_1, c_2 are hyperparameters. Introducing a bonus in the loss function proportional to the policy entropy encourages sufficient exploration during training and prevents early convergence to local minima [23]. An illustrative diagram of the PPO architecture, showing the individual terms of equation (13), is shown in Fig. 2.

Considering the computation of $L^{\text{VF}}(\theta)$, let the n -step return be defined as follows:

$$G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{n-1} r_{t+n} + \gamma^n V_\theta(s_{t+n}) \quad (14)$$

Where V_θ is the estimate of the value function, parameterized by θ . Then, V_t^{target} is defined as the TD(λ) estimator [27]:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t \quad (15)$$

where $\lambda \in [0, 1]$ is a hyperparameter regulating the trade-off between bias and variance in the value function estimation. In particular, values of λ closer to 0 induce higher bias, whereas values of λ close to 1 induce higher variance.

To mitigate sample inefficiency due to being on-policy, PPO can leverage multiple parallel Actors to gather samples from the current policy, which are stored in a buffer and then used to optimize the DNN parameters via minibatch stochastic gradient descent. Unlike in DQN, past experiences are discarded after an update.

As there is no silver bullet in RL, it is difficult to assess a-priori which is the best algorithm for a particular task. For the problem considered in this work, we chose PPO as it was the algorithm that we observed to perform best.

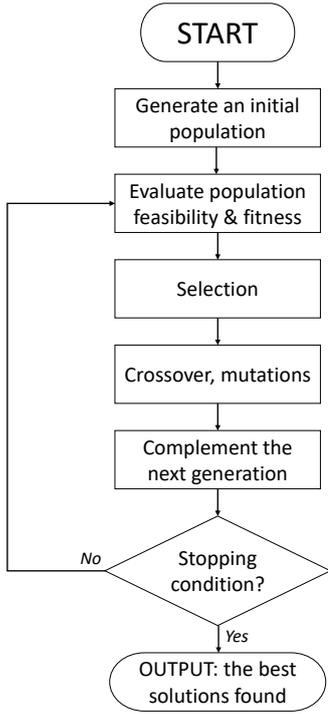


Fig. 3. Genetic Algorithm flow diagram.

C. DRL for Optimization

In [4], methods based on ML for solving traditional optimization problems in Operations Research literature are surveyed. In the framework of optimization algorithms, using a DRL agent, trained to output a feasible solution from an arbitrary problem instance, can be seen as a very complex heuristic, in a framework defined by authors as "end-to-end learning". Heuristic algorithms, in general, can be disassembled into a sequence of subroutines with various degrees of complexity. If one regards the action space of a DRL agent as a collection of subroutines, the task of a DRL agent becomes the one to find a (possibly stochastic) succession of subroutines. Therefore, applying DRL to solving optimization problems can be interpreted as a method of building a specialized heuristic for a certain problem class.

Policy gradient algorithms are able to learn a stochastic policy, which is equivalent to building a heuristic with some stochastic behaviour. We argue that policy stochasticity (i.e., maintaining a small degree of exploration with respect to a deterministic policy) can be a desirable property of DRL when applied to solving optimization problems. In the following, experimental results will confirm the validity of this claim.

IV. ROUTING AND WAVELENGTH ASSIGNMENT

The RWA problem consists in assigning a route and wavelength to a set of lightpath requests in an optical WDM network. The objective here is to maximize the number of accommodated connections. RWA can be either for static traffic (traffic matrix is known in advance), or for dynamic traffic (requests arrive and depart in a stochastic way). This

TABLE I
RWA MODEL PARAMETERS AND VARIABLES

Parameters	
V	Set of routing nodes
E	Set of bidirectional links
W	Set of available wavelengths in each link.
P	Set of pre-computed K-shortest paths.
$P_{(s,d)}$	Set of pre-computed K-shortest paths between node pair (s, d) , $s \in V, d \in V, s \neq d, P_{(s,d)} = K$
$\rho_{(s,d)}$	Number of connections requested by node pair (s, d) , $s \in V, d \in V, s \neq d$
Variables	
x_p^w	1 if wavelength $w \in W$ is occupied in path $p \in P$, 0 otherwise

work considers static RWA, in which we assume wavelength continuity constraints must be enforced for each lightpath.

A. ILP formulation

Static RWA can be formalized as an ILP formulation [34]. ILP parameters and variables are reported in Table I.

$$\max \sum_{p \in P} \sum_{w \in W} x_p^w \quad (16)$$

$$\sum_{p|l \in p} x_p^w \leq 1 \quad \forall l \in E, \forall w \in W \quad (17)$$

$$\sum_{p \in P_{(s,d)}} \sum_{w \in W} x_p^w \leq \rho_{(s,d)} \quad \forall s, d \in V, s \neq d \quad (18)$$

$$x_i^w \in \{0, 1\} \quad \forall i \in P, \forall w \in W \quad (19)$$

Objective function (16) maximizes the number of accommodated connection requests. Constraints (17) impose the wavelength continuity on each established lightpath. Constraints (18) impose that the total accommodated connection requests between each source-destination pair do not exceed the demand. Finally, equations (19) define the variable domains.

Though the ILP provides an exact solution, it is not scalable, as RWA is of NP-Hard complexity [35]. Nevertheless, for small enough problem instances an optimal solution can be found in short computing times, serving as a benchmark for the other algorithms considered in this work.

B. Heuristic and metaheuristic baselines

In the numerical evaluations, the following heuristic and metaheuristic algorithms will be benchmarked against the developed DRL-based solution methods:

- SP-FF: for each connection request, the first wavelength available (i.e., first-fit) in the shortest path between the source-destination pair is selected.
- KSP-FF: for each connection request, the first wavelength available in the shortest among the K-shortest paths between the source-destination pair is selected.
- LLP-FF: for each connection request, the first wavelength available in the least loaded among the K-shortest paths between the source-destination pair is selected.
- GA-FF: a feasible routing is generated via the procedure illustrated in Fig. 3, and the wavelength assignment is performed in a first-fit way.

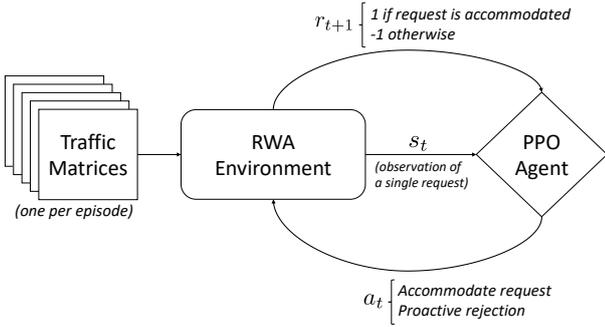


Fig. 4. Illustrative diagram of the learning process. For each episode, the RWA environment draws requests from a traffic matrix sampled from a known probability distribution. Requests are presented to the PPO agent via a properly encoded observation. Based on that information, the PPO agent chooses an action. Finally, the RWA environment produces a reward based on the outcome from the chosen action.

V. DRL ALGORITHM FOR RWA

In this section, our DRL-based approach for solving the RWA problem will be detailed, and the design choices of the developed DRL algorithms will be illustrated.

Given the traffic matrix as input, the trained DRL agent must output a feasible solution for the RWA problem (16)-(19). Therefore, the learning aim is to build a specialized heuristic for the considered RWA problem.

In order to learn a mapping from states to actions, a DRL agent must be provided with proper representations of the environment state, which are called observations. In fact, even though the static RWA problem is a so-called fully-observable environment, providing the agent with the whole traffic matrix and complete information regarding the spectrum occupancy is not an effective strategy. Large observations, other than scalability issues, introduce significant learning challenges, due to the fact that the policy to be learned becomes more complex as the size of an observation increases.

The set of all possible actions a DRL agent can choose from is called action space. Similarly to the previous discussion, an agent with a larger action space indeed has the potential of learning a better policy, however the complexity of the learning phase increases, since more exploration is required.

To summarize, the choice of the state representations (i.e., the observations) and of the action space are of critical importance in designing a DRL algorithm, and it may not be possible a-priori to determine which ones are the most effective for the particular task at hand. Properly shaping the state representation and the expressiveness of the action space, such that the best final performance is attained, constitutes a significant research challenge.

An episode corresponds to a particular RWA problem instance (16)-(19), for which the traffic matrix is drawn from a given probability distribution. For the problem instances used for training, the traffic distribution is assumed to be uniform for each possible source-destination pair. During an episode, at each time step, the agent is presented with a single connection request and it has to decide, based on an observation, whether to try to accommodate it or to proactively

reject it. Therefore, the number of time-steps in an episode is equal to the number of connection requests. At the end of an episode, the allocated spectral resources are freed, and a new traffic matrix is processed. An illustrative diagram of the learning framework is shown in Fig. 4.

Observations represent the network state by including the following information: the source-destination pair for the request to be accommodated at the current time-step, and the spectrum occupation in each of the K -shortest candidate paths. Even though there is some loss of information with respect to the full traffic matrix and the complete spectrum occupation in the network, experimental results suggest that it is sufficient for the agents to learn an effective policy.

We train two distinct agents, differing only in how the action space and the observation space were defined, to assess the impact of these design choices on performance.

We use PPO [33] to train the agents, given that it is one of the main state-of-the-art DRL algorithms. The agents are trained via the PPO algorithm implementation in Stable Baselines, an open-source library providing reliable implementations of state-of-the-art DRL algorithms [36].

A. Agent 1: PPO-FF

The goal of PPO-FF is to learn a stochastic, first-fit policy given information regarding the current request and the spectrum occupancy of the K -shortest paths between request's source and destination nodes.

1) *Action space*: The action space for this agent is discrete and of dimension $(K + 1)$, i.e., each action corresponds either to selecting one of the K -shortest paths for the source-destination pair of the current request, or to a proactive rejection. When a path is selected, if there is at least one available wavelength along the selected path, the first wavelength is used to accommodate the request. Otherwise, if no wavelengths are available, the request is rejected.

2) *Observations*: An observation consists in a $(2|V| + 2K)$ vector. In particular, a $(2|V|)$ vector contains the one-hot-coded source-destination pair for the current request, a K -dimensional binary vector determines whether or not there is a wavelength available in each of the K paths, and a K -dimensional vector contains the total load on each path. Observations are normalized to be in the range $[-1, 1]$.

B. Agent 2: PPO-Full

The goal of PPO-Full is to learn a stochastic policy given information regarding the current request and the spectrum occupancy of the K -shortest paths. Differently from PPO-FF, PPO-Full can accommodate a request in any wavelength available in each of the K paths.

1) *Action space*: The action space for this agent is discrete and of dimension $(K|W| + 1)$, such that each action corresponds either to selecting a path and a wavelength or to a proactive rejection. If a path and a wavelength are selected, the current request is accommodated to the selected path and to the selected wavelength, if available. Otherwise, if no wavelengths are available, the request is rejected.

2) *Observations*: An observation consists in a $(2|V| + K(|W| + 1))$ vector. In particular, a $(2|V|)$ vector contains the one-hot coded source and destination nodes for the current request, a $(K|W|)$ binary matrix determines the wavelength availability in each of the K paths, and a K -dimensional vector contains the total load in each of the K paths. Observations are normalized to be in the range $[-1, 1]$.

Compared to PPO-FF, PPO-Full has a larger action space, which grants it more freedom in deciding how to accommodate requests. However, larger action space introduce more difficult learning challenges, both in terms of exploration of the action space and in terms of having to learn a more complex mapping from observations to actions.

C. Reward Function

We have defined two different reward functions for the agents: a sparse reward and a shaped reward. The sparse reward reads as follows:

$$r_{\text{sparse}} = \begin{cases} 1 & \text{if accommodated} \\ -1 & \text{otherwise} \end{cases} \quad (20)$$

The shaped reward reads as follows:

$$r_{\text{shaped}} = \begin{cases} \alpha \frac{C_j}{\max_i C_i} + \beta \frac{\min_i h_i}{h_j} & \text{if accommodated} \\ 0 & \text{otherwise} \end{cases} \quad (21)$$

where $i = 1 \dots K$, K is the number of shortest paths for each source-destination pair, j is the chosen path among the K -shortest, C_i is the capacity (i.e., the number of available wavelengths) on path i , h_i are the number of hops taken by path i , and α, β are positive scalars. The shaped reward induces the agent to find a trade-off between a load balancing strategy and a shortest-path strategy.

The sparse reward holds more expressive power with respect to the shaped reward, as it does not bias the agent with any specific strategy. However, when the link capacity is high, using the sparse reward results in all actions (except proactive rejection) having reward equal to +1 for the majority of a training episode, being the network unloaded. For low link capacities, we have observed from experimental results that using the shaped rewards yields performance worse or comparable performance compared to using the sparse reward. Therefore, in the following, we will employ the shaped reward only in the case of high link capacities. Overall, the use of a carefully shaped reward is suggested only if learning using a sparse reward is exceedingly difficult.

D. Multi-Start Agents

Both PPO-FF and PPO-Full learn a stochastic policy: given an observation, the agents output a selection probability for each action in the action space. Therefore, it is possible to evaluate the agents in either a deterministic or a non-deterministic way. With a deterministic evaluation, the agent always chooses the action with the highest selection probability. With a non-deterministic evaluation, the agent samples an action based on the selection probabilities, therefore maintaining a degree of exploration. While a deterministic evaluation is paramount

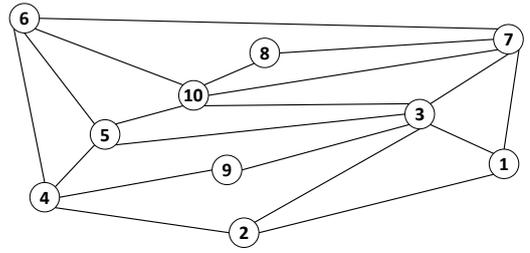


Fig. 5. 10-node reference network used for training the agents.

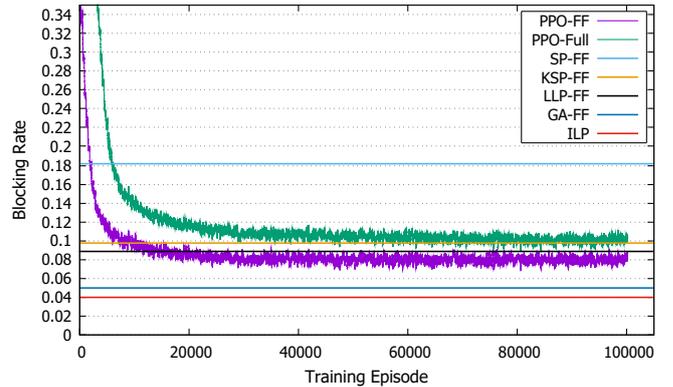


Fig. 6. Blocking rate of the PPO agents as a function of the training episode. The average blocking rates over 1000 test traffic matrices of the greedy heuristics, Genetic Algorithm and ILP are also reported.

for assessing the performance of an agent in applications such as real-time control, we note that non-determinism is a desirable property when applying DRL to solving optimization problems. Leveraging on policy stochasticity draws a parallelism with metaheuristics that implement random behaviours in order to escape local minima. For this reason, the average performance of an agent results to be an underestimation of what can actually be achieved.

To take advantage from policy stochasticity, a trained DRL agent can be cast within a Multi-Start (MS) heuristic framework [37], in which the same problem instance is solved multiple times and the only best solution is retained. Clearly, this approach introduces a significant computational overhead, as it evaluates the same RWA environment multiple times, but better solutions can be achieved with respect to a deterministic evaluation. In our evaluations, we chose a number of independent starts equal to 8 (MSx8), as lower numbers would not bring a significant improvement with respect to a deterministic evaluation, and higher numbers would not bring any significant further improvements. In the following, we will denote PPO-FF-MSx8 and PPO-Full-MSx8 the Multi-Start versions of PPO-FF and PPO-Full, respectively.

We note that casting DRL agents in a Multi-Start framework has already been explored in [38], albeit for a completely different application. To the best of our knowledge, this is the first work that applies Multi-Start in DRL for optimization in optical networks.

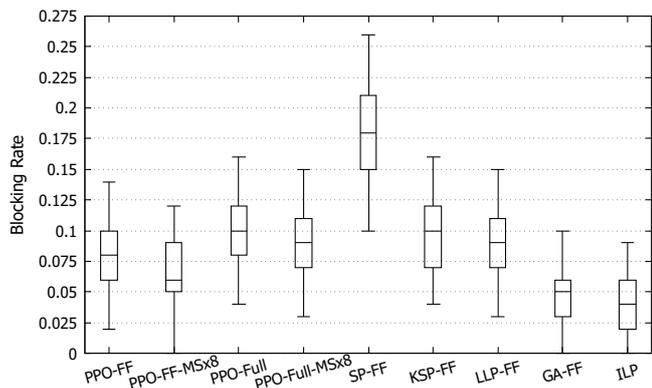


Fig. 7. Blocking rate of the PPO agents, heuristics and ILP on 1000 traffic matrices from the training environment. Whiskers encompass 95% of the data.

VI. ILLUSTRATIVE NUMERICAL RESULTS

In this section, we compare the proposed DRL-based approaches to 1) baseline greedy heuristics, 2) GA (as representative of a state-of-the-art metaheuristic) and 3) ILP, in terms of request blocking rate and computational time.

The implementation for the RWA environment is based on the Optical RL-Gym framework [39]. The baseline greedy heuristics considered in this work are Shortest-Path-First-Fit (SP-FF), K-Shortest-Path-First-Fit (KSP-FF) and Least-Loaded-Path-First-Fit (LLP-FF). A flow diagram describing the behaviour of the Genetic Algorithm (GA-FF) is illustrated in Fig. 3 [40], [41]. The parameters of GA-FF have been fine-tuned in order to get the best trade-off between computing times and solution quality. Finally, exact solutions were obtained by solving the ILP formulation (16)-(19) via the state-of-the-art commercial solver Gurobi v.9.1 [42].

The reference 10-node network used for training and evaluation is illustrated in Fig. 5. The number of pre-computed K-shortest paths was set to $K = 3$, and path lengths were calculated in terms of number of hops. The number of wavelengths available in each link was set to $W = 10$. The agents were trained for a total of 10^6 episodes, with each episode consisting of 100 connection requests. In particular, requests were generated according to a uniform traffic distribution between all the possible source-destination pairs.

For the hyperparameters of PPO, we chose a shared architecture for the policy and the value function. We chose a Multi-Layer Perceptron (MLP) of 5 layers with 128 neurons each, with Exponential Linear Units (ELU) as activation functions. We have set the discount factor γ to 0.95, the learning rate to $5 \cdot 10^{-5}$, the entropy coefficient c_2 to 10^{-4} , the buffer size to 32768, and the minibatch size to 1024. All the other hyperparameters were left as their default values. For a detailed investigation regarding efficient training of on-policy algorithms, in particular PPO, the reader can refer to [43].

A. Performance on the training environment

Fig. 6 shows the blocking rates of the two agents during the training phase, and the average blocking rates over 1000 test problem instances computed by the considered heuristic

algorithms and the ILP model PPO-Full, even though it has more freedom of action than PPO-FF, achieves worse average performance with respect to PPO-FF at the end of the training phase. This behaviour is due to the fact that PPO-Full must deal with a larger action space and more complex observations than PPO-FF, therefore the learning policy becomes more complex. In fact, PPO-Full outperforms only SP-FF, and, on average, it has worse performance than all the other considered heuristic algorithms. On the other hand, PPO-FF is able to outperform, on average, all the considered baseline greedy heuristics, with 57.3%, 19.8% and 13.4% improvement with respect to SP-FF, KSP-FF and LLP-FF, respectively. However, the 36.1% gap with respect to GA-FF shows that PPO-FF on average does not outperform a fine-tuned metaheuristic. Finally, a significant 48.9% optimality gap between PPO-FF and ILP suggests that there is still much room for improvement.

In Fig. 7 the blocking rates of the PPO agents, heuristics and ILP are reported, evaluated on 1000 traffic matrices drawn from the training environment. We observe that Multi-Start can indeed improve the performance of PPO agents, as policy stochasticity is an effective tool for escaping local optima. Moreover, even though GA-FF is still the best heuristic, PPO-FF-MSx8 is able to close the gap, much more effectively than PPO-FF. This is a remarkable result, given the simplicity of the action space of PPO-FF-MSx8 with respect to a sophisticated metaheuristic such as GA-FF.

B. Generalization capabilities

Due to overfitting, the trained DRL agents may experience performance degradation when evaluated on traffic matrices that differ in distribution from the ones generated in the training phase. Therefore, since realistic application scenarios may sensibly differ from the training environment, properly assessing the generalization capabilities of the trained DRL agents is paramount. In the following, the generalization capabilities of the trained DRL agents will be assessed on traffic matrices unseen during training, in terms of the total number of connection requests and of the traffic distribution.

1) *Generalization on the total number of requests:* To assess the generalization capability of our DRL agents to a different number of connection requests per episode, we consider the cases when the performance of the DRL agents are tested with 100, 105, 110, 125 and 150 connection requests per episode. We remark that, in all the above mentioned cases, the DRL agents were always trained considering a total of 100 connection requests per training episode, and that the traffic distribution is assumed to be uniform between all possible source-destination pairs for all requests. The average blocking rates of the DRL agents, of the heuristics and of the ILP for different numbers of total requests are reported in Fig. 8a.

Comparing the relative performance of the DRL agents with the other approaches, it can be observed that the PPO-FF and PPO-Full suffer some slight performance degradation as the number of total connection requests increases. This suggests a small degree of overfitting with respect to the number of requests seen during training. In particular, for 150 total connection requests (i.e., a 50% increase with respect to training), PPO-FF is on-par with KSP-FF.

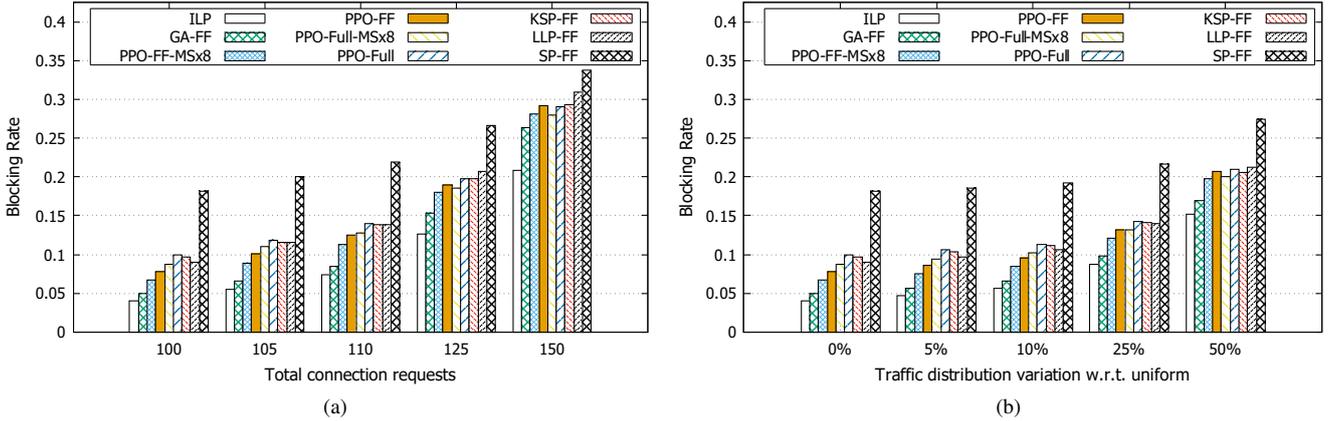


Fig. 8. Average blocking rates of the PPO agents, heuristics and ILP for 1000 test traffic matrices, as a function of: (a) the total number of connection requests, and (b) the variation from the training uniform traffic distribution.

On the other hand, GA is able to consistently outperform PPO-FF by an average of 26.4%. This is a reasonable result, given that GA runs more complex subroutines (i.e., crossover, mutations, selection) in order to generate a feasible solution. Moreover, GA went through an extensive process of parameter fine-tuning to get the possible best blocking rates. For this particular task, GA has proven to be extremely sensitive to parameter tuning, to the point that a wrong choice in the parameters would have resulted in a solution worse than the greedy heuristics. PPO, on the other hand, has proven to be quite robust to the choice of hyperparameters.

Since the observations do not depend on the number of total connection requests, it is reasonable to assume that the agents were able to learn a policy that can generalize with respect to the number of requests per episode. Furthermore, the best DRL agent is not unequivocal, and the performance gap between PPO-FF and PPO-Full becomes smaller with increasing total number of requests per episode, suggesting that for a high traffic load there may not be a significant benefit in employing either of the two strategies. However, PPO-FF exhibits better performance overall with respect to PPO-Full, and works with a simpler observation space. Therefore, since the policy to be learned is less complex, fine-tuning the DNN architecture may further lighten the computational burden of the DRL agent.

The solutions computed by the DRL agents can be further improved by leveraging stochasticity and employing a Multi-Start framework. Considering PPO-FF, employing Multi-Start with 8 independent starts (PPO-FF-MSx8) allows to improve performance by an average of 8.77%, with a best improvement of 13.8% for 100 total connection requests. The relative improvement provided by the Multi-Start becomes less significant as the load increases, as the average absolute improvement in the blocking rate is equal to 1.11%, which becomes negligible for higher blocking rates. For instance, with 150 total connection requests, only a 3.7% relative improvement is gained by the Multi-Start. PPO-FF-MSx8 brings an average improvement of 43.9%, 17.0% and 17.7% over SP-FF, KSP-FF and LLP-FF, respectively, thus surpassing by a significant margin all baseline greedy heuristics. Furthermore, PPO-FF-

MSx8 is able to improve the gap with respect to the near-optimal performance provided by GA, with an average gap of 19.7%.

2) *Generalization on the traffic distribution:* We now evaluate how the performance of the DRL agents is affected when the traffic distribution considered during test phase is different compared from the one used during DRL algorithm training. To do so, during the test phase we do not consider that all source-destination pairs are chosen with uniform probability, but alter their probability considering different variations, namely $\pm 5\%$, $\pm 10\%$, $\pm 25\%$ and $\pm 50\%$ with respect to the uniform distribution. The total number of connection requests was set to 100 for all instances. The average blocking rates of the DRL agents, of the heuristics and of the ILP for the different traffic distributions are reported in Fig. 8b.

In general, the blocking rates increase with the amount of variation from the uniform distribution. Indeed, by keeping the total number of requests the same, an unbalanced distribution is more likely to create situations of congestion, given that more spectral resources are requested from specific source-destination pairs, leading to overutilization of specific links.

As in the previous evaluation, by comparing the relative performance of the DRL agents with the other solution methods, it can be observed that PPO-FF and PPO-Full do suffer from performance degradation when tested on traffic distributions different than the ones seen during training. This suggests a degree of overfitting with respect to the traffic distribution seen during training. On the extreme case, it can be seen that PPO-FF shows similar performance to KSP-FF for a 50% variation with respect to the training distribution. On the other hand, PPO-FF is able to outperform all the baseline greedy heuristics for all the other considered sets of instances, which are more similar to the train distribution. Furthermore, for the same reasons commented in the previous subsection, the gap in performance between PPO-FF and PPO-Full becomes smaller as the tested distribution becomes more and more unbalanced.

As seen before, DRL agents can improve the solutions using Multi-Start. In particular, considering PPO-FF, employing Multi-Start with 8 independent starts (PPO-FF-MSx8) allows

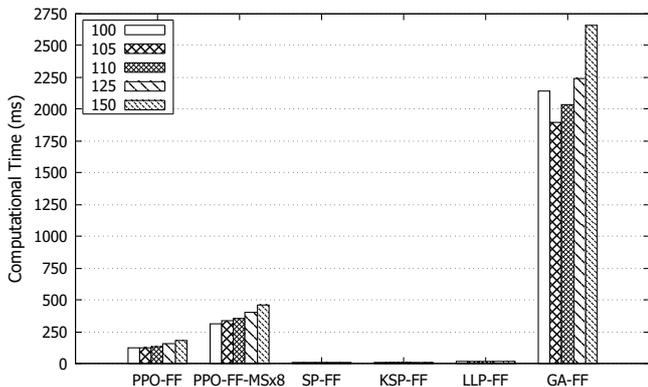


Fig. 9. Average computational times of the DRL agents and the heuristics varying the number of connection requests.

to improve performance by an average of 10.1%, and the absolute improvement provided by the Multi-Start becomes less significant as the distribution becomes more unbalanced, i.e., when the network congestion increases. PPO-FF-MSx8 is able to consistently outperform all greedy heuristics, with an average improvement of 50.1%, 19.9% and 17.6% for SP-FF, KSP-FF and LLP-FF respectively. Similarly to the previous case, GA-FF is able to consistently outperform PPO-FF, with an average gap of 28.9%. With PPO-FF-MSx8, it is possible to reduce this gap, reaching an average 21.0% gap, at the price of a higher computational overhead due to the Multi-Start.

C. Computational times

Fig. 9 reports the average computational times required by the DRL agents and the heuristics varying the number of total number of connection requests¹. For the DRL agents, only PPO-FF is reported, since the computational times of PPO-Full are very similar, given that they employ the same DNN architecture. The greedy heuristics are by far the fastest approaches, as expected. PPO-FF-MSx8 is on average 61.5% times slower than PPO-FF, given that it requires eight independent executions of PPO-FF. The additional computational overhead by the Multi-Start is mitigated via the use of multiprocessing on each independent start. GA-FF is by far the slowest method, more than five times slower than PPO-FF-MSx8, and an order of magnitude slower than PPO-FF.

Note that the parameters of GA-FF could be tuned in order to reduce the computing times, but that would have impacted severely the solution quality, to the point of being either on-par or plainly outperformed by PPO-FF and PPO-FF-MSx8. Therefore, to get the best solution quality possible out of GA-FF, one must tolerate relatively long execution times compared to the other solution methods.

D. Scalability

To assess the scalability of the proposed DRL-based approach, the two following scenarios will be considered: 1)

¹GA-FF computational times may not increase monotonically due to several early stopping conditions, whose effect on the computational times is unpredictable. Early stopping avoids excessive computing times while preserving solution quality.

networks with a larger number of nodes, and 2) networks with a higher number of wavelengths per link. In the following, only PPO-FF and PPO-FF-MSx8 are considered, as PPO-Full and PPO-Full-MSx8 were shown to exhibit an overall worse performance.

1) *Larger networks*: In this evaluation setting, the 24-node US backbone topology and a 100-node Gabriel graph [44] were considered. Because of their grid-like structure, Gabriel graphs accurately mimic the graph structure of physical layer topologies [45]. For each topology, the number of wavelengths per link was set to 10 and the number of connection requests per episode, uniformly distributed among all source-destination pairs, was set to 100.

Fig. 10a shows the blocking rates for the PPO agent, the baseline heuristics and ILP for the 24-node US backbone and the 100-node Gabriel graph. For both networks, PPO-FF learns a policy that outperforms SP-FF and KSP-FF, but attains very similar results with respect to LLP-FF. With the use of Multi-Start, PPO-FF-MSx8 is able to outperform all greedy heuristics. As for the 10-node network, there is still a gap in performance with respect to GA-FF and ILP.

2) *Higher capacity networks*: In this evaluation setting, the 10-node network in Fig. 5, the 24-node US backbone and a 100-node Gabriel graph were considered. For each topology, the number of wavelengths per link was set to 80 and the number of connection requests per episode, uniformly distributed among all source-destination pairs, was set to 800. For this scenario, we used the shaped reward defined in Eq. (21). According to [46], and confirmed by preliminary experiments, a policy that favours the least loaded choice is more effective when there is blocking in the network. Therefore, the values for α and β in Eq. (21) have been set to $\alpha = 0.9$ and $\beta = 0.1$.

Moreover, an additional element was added to the state observation, namely, a (K) vector containing the length in hops for each of the candidate paths.

Fig. 10b shows the blocking rates for the PPO agent, the baseline heuristics for the 10-node network, the 24-node US backbone and the 100-node Gabriel graph. ILP results for the 100-node network are not reported due to excessively long computational times. For the 10-node network, using the shaped reward, both PPO-FF and PPO-FF-MSx8 are able to outperform the greedy heuristics and attain comparable performance with respect to GA-FF. For the 24-node network, only PPO-FF-MSx8 is able to outperform all greedy heuristics, in particular slightly outperforming LLP-FF and attaining comparable performance with respect to GA-FF. Finally, for the 100-node network, PPO-FF and PPO-FF-MSx8 are able to outperform SP and KSP-FF, but do not outperform LLP-FF. This suboptimal performance of PPO-FF and PPO-FF-MSx8 for the largest instance suggests a need for more sophisticated DRL-based solution methods than one which only learns an admission control and routing rule.

VII. CONCLUSION

In this work, we compared the performance of DRL-based methods against a state-of-the-art metaheuristic and ILP for

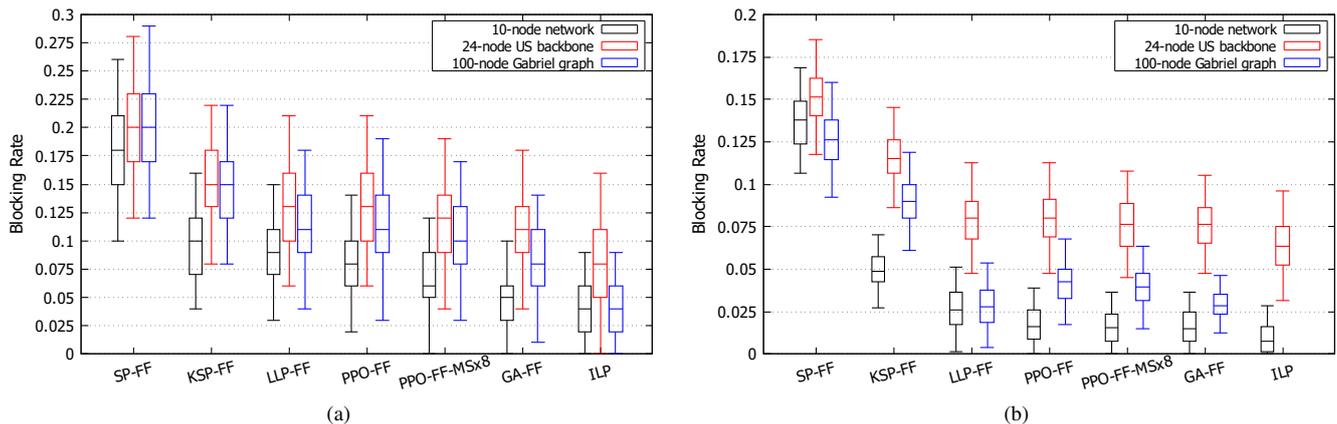


Fig. 10. Blocking rates of the PPO agents, heuristics and ILP for 1000 test traffic matrices. (a) 10 wavelengths per link, 100 uniformly distributed connection requests per episode. (b) 80 wavelengths per link, 800 uniformly distributed connection requests per episode. Whiskers encompass 95% of the data.

solving the static RWA problem. We trained and evaluated two DRL agents, namely PPO-FF and PPO-Full. In particular, PPO-FF applies a first-fit strategy for wavelength assignment, whereas PPO-Full can accommodate requests in any available wavelength. Furthermore, by leveraging stochasticity in the learned policy, we developed a Multi-Start approach (PPO-FF-MSx8 and PPO-Full-MSx8), bringing a consistent improvement on the average performance of the DRL agents. Finally, we engineered a shaped reward that allows efficient learning in networks with high link capacity. From experimental results, we observed that PPO-FF outperforms PPO-Full in most cases, highlighting the significant challenges in terms of exploration and feature learning in DRL. Overall, our proposed approach shows pros and cons with respect to both greedy heuristics (i.e., longer computational times, but better solution quality) and state-of-the-art metaheuristics such as GA-FF (i.e., worse solution quality, but better computational times), providing a competitive middle ground between these two aspects. Future work will focus on learning more complex heuristic frameworks (e.g., local search) leveraging DRL-based methods.

ACKNOWLEDGMENT

The authors would like to thank Lily Friedberg for the insightful discussions about the paper.

REFERENCES

- [1] C. Rottondi, L. Barletta, A. Giusti, and M. Tornatore, "Machine-learning method for quality of transmission prediction of unestablished light-paths," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, no. 2, pp. A286–A297, 2018.
- [2] F. Musumeci, C. Rottondi, G. Corani, S. Shahkarami, F. Cugini, and M. Tornatore, "A tutorial on machine learning for failure management in optical networks," *Journal of Lightwave Technology*, vol. 37, no. 16, pp. 4125–4139, 2019.
- [3] S. Troia, R. Alvizu, Y. Zhou, G. Maier, and A. Pattavina, "Deep learning-based traffic prediction for network optimization," in *2018 20th International Conference on Transparent Optical Networks (ICTON)*, 2018, pp. 1–4.
- [4] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.
- [5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.
- [6] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.
- [7] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," 2019. [Online]. Available: <https://arxiv.org/abs/1912.06680>
- [8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.
- [9] Y. Duan, X. Chen, R. Houthoof, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48, Jun 2016, pp. 1329–1338.
- [10] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *CoRR*, vol. abs/1709.07080, 2017. [Online]. Available: <http://arxiv.org/abs/1709.07080>
- [11] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep reinforcement learning meets graph neural networks: An optical network routing use case," *CoRR*, vol. abs/1910.07421, 2019. [Online]. Available: <http://arxiv.org/abs/1910.07421>
- [12] X. Chen, B. Li, R. Proietti, H. Lu, Z. Zhu, and S. J. B. Yoo, "Deepprmsa: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks," *Journal of Lightwave Technology*, vol. 37, no. 16, pp. 4155–4163, 2019.
- [13] B. Li and Z. Zhu, "Deepcoop: Leveraging cooperative drl agents to achieve scalable network automation for multi-domain sd-eons," in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, 2020, pp. 1–3.
- [14] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," 2019.
- [15] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," *CoRR*, vol. abs/1812.02341, 2018. [Online]. Available: <http://arxiv.org/abs/1812.02341>
- [16] K. Wang, B. Kang, J. Shao, and J. Feng, "Improving generalization in reinforcement learning with mixture regularization," in *NeurIPS*, 2020. [Online]. Available: <https://arxiv.org/pdf/2010.10814.pdf>
- [17] R. Ramaswami and K. Sivarajan, "Routing and wavelength assignment in all-optical networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 5, pp. 489–500, 1995.
- [18] K. Christodoulopoulos, K. Manousakis, and E. Varvarigos, "Offline

- routing and wavelength assignment in transparent wdm networks,” *IEEE/ACM Transactions on Networking*, vol. 18, no. 5, pp. 1557–1570, 2010.
- [19] X. Chu and B. Li, “Dynamic routing and wavelength assignment in the presence of wavelength conversion for all-optical networks,” *IEEE/ACM Transactions on Networking*, vol. 13, no. 3, pp. 704–715, 2005.
- [20] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, “Applications of deep reinforcement learning in communications and networking: A survey,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.
- [21] J. Suarez-Varela, A. Mestres, J. Yu, L. Kuang, H. Feng, A. Cabellos-Aparicio, and P. Barlet-Ros, “Routing in optical transport networks with deep reinforcement learning,” *Journal of Optical Communications and Networking*, vol. 11, no. 11, pp. 547–558, 2019.
- [22] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.
- [23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: <http://arxiv.org/abs/1602.01783>
- [24] X. Chen, R. Proietti, C.-Y. Liu, and S. J. B. Yoo, “A multi-task-learning-based transfer deep reinforcement learning design for autonomic optical networks,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 9, pp. 2878–2889, 2021.
- [25] X. Chen, R. Proietti, and S. J. B. Yoo, “Building autonomic elastic optical networks with deep reinforcement learning,” *IEEE Communications Magazine*, vol. 57, no. 10, pp. 20–26, 2019.
- [26] M. R. Raza, C. Natalino, P. Ohlen, L. Wosinska, and P. Monti, “Reinforcement learning for slicing in a 5g flexible ran,” *Journal of Lightwave Technology*, vol. 37, no. 20, pp. 5161–5169, 2019.
- [27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: <http://incompleteideas.net/book/the-book-2nd.html>
- [28] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” 2015.
- [29] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling network architectures for deep reinforcement learning,” 2016.
- [30] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” 2017.
- [31] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.
- [32] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” 2018.
- [33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: <http://arxiv.org/abs/1707.06347>
- [34] A. Ozdaglar and D. Bertsekas, “Routing and wavelength assignment in optical networks,” *IEEE/ACM Transactions on Networking*, vol. 11, pp. 259–272, 01 2003.
- [35] I. Chlamtac, A. Ganz, and G. Karmi, “Lightpath communications: an approach to high bandwidth optical wan’s,” *IEEE Transactions on Communications*, vol. 40, no. 7, pp. 1171–1182, 1992.
- [36] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: <http://jmlr.org/papers/v22/20-1364.html>
- [37] R. Martí, *Multi-Start Methods*. Boston, MA: Springer US, 2003, pp. 355–368.
- [38] V. Antuori, E. Hebrard, M.-J. Huguet, S. Essodaigui, and A. Nguyen, “Leveraging reinforcement learning, constraint programming and local search: A case study in car manufacturing,” in *International Conference on Principles and Practice of Constraint Programming*, 2020.
- [39] C. Natalino and P. Monti, “The Optical RL-Gym: an open-source toolkit for applying reinforcement learning in optical networks,” in *International Conference on Transparent Optical Networks (ICTON)*, July 2020, p. Mo.C1.1. [Online]. Available: <https://github.com/carlosnatalino/optical-rl-gym>
- [40] O. Karandin, F. Musumeci, O. Ayoub, A. Ferrari, Y. Pointurier, and M. Tornatore, “Quantifying resource savings from low-margin design in optical networks with probabilistic constellation shaping,” in *47th European Conference on Optical Communication (ECOC 2021)*, September 2021.
- [41] M. Ibrahim, O. Ayoub, O. Karandin, F. Musumeci, A. Castoldi, R. Pastorelli, and M. Tornatore, “Qot-aware optical amplifier placement in filterless metro networks,” *IEEE Communications Letters*, vol. 25, no. 3, pp. 931–935, 2021.
- [42] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2021. [Online]. Available: <https://www.gurobi.com>
- [43] M. Andrychowicz, A. Raichuk, P. Stanczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, “What matters in on-policy reinforcement learning? A large-scale empirical study,” *CoRR*, vol. abs/2006.05990, 2020. [Online]. Available: <https://arxiv.org/abs/2006.05990>
- [44] K. R. Gabriel and R. R. Sokal, “A New Statistical Approach to Geographic Variation Analysis,” *Systematic Biology*, vol. 18, no. 3, pp. 259–278, 09 1969.
- [45] E. K. Çetinkaya, M. J. Alenazi, Y. Cheng, A. M. Peck, and J. P. Sterbenz, “On the fitness of geographic graph generators for modelling physical level topologies,” in *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2013, pp. 38–45.
- [46] R. Romero Reyes and T. Bauschert, “Towards drl-based routing and spectrum assignment in optical networks: Lessons to be learned from markov decision processes,” in *2021 IEEE Latin-American Conference on Communications (LATINCOM)*, 2021, pp. 1–6.

Nicola Di Cicco (S’22) is a Ph.D. Student with the Department of Electronic, Information and Bioengineering (DEIB) at Politecnico di Milano, Milan, Italy. His current research interests are in the field of deep learning and reinforcement learning for network optimization and automation.

Emre Furkan Mercan is currently a Field Test Engineer at umlaut, part of Accenture, in Aachen, Germany. He holds a M.Sc. degree in Telecommunications Engineering from Politecnico di Milano. His current research interests are in the field of communication networks optimization, with an emphasis on 5G NR network optimization and testing.

Oleg Karandin is a Ph.D. Student with the Department of Electronic, Information and Bioengineering (DEIB) at Politecnico di Milano, Milan, Italy. His current research interests are in the field of low-margin optical network design and machine learning for Quality-of-Transmission estimation.

Omran Ayoub is currently a researcher with the Department of Innovative Technologies (DTI) at Scuola Universitaria Professionale della Svizzera Italiana. His current research interests are in the field of network optimization and artificial intelligence and machine learning for communication networks.

Sebastian Troia is an Assistant Professor with the Department of Electronics, Information and Bioengineering (DEIB) at Politecnico di Milano, Milan, Italy. His current research interests are in the field of edge network softwareization and machine-learning for SDN and SD-WAN based networks.

Francesco Musumeci (S’11-M’12) is Assistant Professor with the Department of Electronics, Information and Bioengineering at Politecnico di Milano. His current research interests include Machine-Learning-assisted networking, design and optimization of optical networks, and network disaster resilience.

Massimo Tornatore (S’03–M’06–SM’13) is currently an Associate Professor at Politecnico di Milano. He also holds an appointment as Adjunct Professor at University of California, Davis, USA and as visiting professor at University of Waterloo, Canada. His research interests include performance evaluation, optimization and design of communication networks (with an emphasis on the application of optical networking technologies), cloud computing, and machine learning application for network management. In these areas, he co-authored more than 400 peer-reviewed conference and journal papers (with 19 best paper awards), 2 books and 1 patent. He is a member of the Editorial Board of IEEE Communication Surveys and Tutorials, IEEE Communication Letters, Springer Photonic Network Communications, and Elsevier Optical Switching and Networking.

3.2. Reinforcement-Learning-based Local Search for Network Optimization

The main conclusion drawn from the previous paper is that DRL alone is unlikely to learn better solving algorithms for network optimization problems than conventional approaches from Operations Research. Motivated by these results, this paper takes on a different idea. The performance of complex metaheuristics heavily relies on the design of their subroutines, e.g., the neighborhood selection policy in Local Search or the mutation and crossover functions in Genetic Algorithms. While, traditionally, these subroutines are fixed among problems, in this paper, we explore the idea of learning specialized subroutines for network optimization problems via DRL. In particular, we consider a simple Local Search algorithm, and we substitute its neighborhood selection policy with a DRL agent, resulting in the DeepLS algorithm. The idea of DeepLS is very simple: given a neighbourhood, a neural network ingests the state of the optimization problem (comprising current and best values of the decision variables alongside problem-specific features). Based on this information, the neural network predicts the best variables to perturb to achieve the best objective value in the long term. To enable scalability and generalization, we leverage a special type of neural networks, namely, equivariant neural networks, such that DeepLS can be applied to networks larger than the ones seen during training without the need for retraining or fine-tuning. As representative scenarios, we apply DeepLS to two classical hard combinatorial optimization problems in networking, namely, Routing and Wavelength Assignment in optical networks, and traffic engineering with OSPF in IP networks. Our numerical results are promising, with DeepLS outperforming state-of-the-art metaheuristics and learning-based solving methods in multiple optimization problems, network topologies, and traffic distributions. Overall, we conclude that integrating lightweight DRL submodules into larger-scope OR algorithms is a promising path toward practical learning-based network optimization algorithms.

DeepLS: Local Search for Network Optimization based on Lightweight Deep Reinforcement Learning

Nicola Di Cicco, *Graduate Student Member, IEEE*, Mamedhe Ibrahim, *Member, IEEE*
 Sebastian Troia, *Member, IEEE*, and Massimo Tornatore, *Fellow, IEEE*

Abstract—Deep Reinforcement Learning (DRL) is being investigated as a competitive alternative to traditional techniques for solving network optimization problems. A promising research direction lies in enhancing traditional optimization algorithms by offloading low-level decisions to a DRL agent. In this study, we consider how to effectively employ DRL to improve the performance of Local Search algorithms, i.e., algorithms that, starting from a candidate solution, explore the solution space by iteratively applying local changes (i.e., moves), yielding the best solution found in the process. We propose a Local Search algorithm based on lightweight Deep Reinforcement Learning (DeepLS) that, given a neighborhood, queries a DRL agent for choosing a move, with the goal of achieving the best objective value in the long term. Our DRL agent, based on permutation-equivariant neural networks, is composed by less than a hundred parameters, requiring only up to ten minutes of training and can evaluate problem instances of arbitrary size, generalizing to networks and traffic distributions unseen during training. We evaluate DeepLS on two illustrative NP-Hard network routing problems, namely OSPF Weight Setting and Routing and Wavelength Assignment, training on a single small network only and evaluating on instances 2x-10x larger than training. Experimental results show that DeepLS outperforms existing DRL-based approaches from literature and attains competitive results with state-of-the-art metaheuristics, with computing times up to 8x smaller than the strongest algorithmic baselines.

Index Terms—Deep Reinforcement Learning, Local Search, IP Networks, Optical Networks, NP-Hard Optimization

I. INTRODUCTION

Machine Learning (ML) has become a widely adopted decision-support tool in communications networks. Remarkable results have been achieved in many applications, such as in traffic prediction [1], failure detection [2], traffic engineering [3], link load control [4], and routing and spectrum assignment problems in optical networks [5], [6].

As the size and the complexity of network optimization problems are bound to grow over time, new solving methods that are both computationally scalable and that can produce good-quality solutions are currently sought for several networking domains, such as in IP and in optical networks. Among the different ML paradigms, Reinforcement Learning (RL) is currently attracting significant attention. What makes RL particularly attractive is its capability, unlike Supervised Learning (SL), to learn without labelled data. The high-level objective of RL applied to NP-Hard optimization problems is to learn specialized heuristic algorithms tailored for specific

problem classes (e.g., learning a constructive heuristic for static Routing and Wavelength Assignment (RWA) in optical networks [6]). Following this line of reasoning, RL-based heuristics have been applied with success to several classical problems in the Operations Research (OR) literature, such as the Knapsack Problem, the Bin-Packing Problem, the Traveling Salesman Problem, and the Capacitated Facility Location Problem [7]–[10].

Generally speaking, Bengio et al. [11] have laid out three high-level paradigms for ML applied to combinatorial optimization: *i)* end-to-end learning, *ii)* parameter configuration, and *iii)* ML alongside OR. In end-to-end learning, a ML black-box model outputs a feasible solution for a given problem instance. In parameter configuration, a ML model predicts the optimal setup parameters of a given OR algorithm, for instance a solver or a metaheuristic, for a given problem instance. Finally, in ML alongside OR, one or more ML-based algorithms replace lower-level functions that were originally implemented by an OR algorithm (e.g., choosing the best variable to branch in each Branch-and-Bound iteration [10]).

Among the aforementioned paradigms, ML alongside OR (ML+OR) stands out, as it preserves the logical structure of well-established optimization strategies, while leveraging ML to address their shortcomings. For example, many low-level components in well-known heuristics are often arbitrarily defined, such as the criteria for choosing a move in Local Search or the mutation/crossover functions in a Genetic Algorithm. As an illustrative recent research achievement, the winner of a recent competition on solving the Vehicle Routing Problem with Time Windows [12] employed ML+OR methodologies. Therefore, we argue that investigating ML+OR methodologies for solving classical optimization problems in networking is a research topic of great interest. Our research questions are therefore as follows: *can we enhance already existing network optimization algorithms by offloading selected tasks to (possibly lightweight) learning intelligence? What are the gains compared with handcrafted state-of-the-art algorithms?*

To demonstrate the potential of ML+OR for network optimization, we propose DeepLS, a simple Local Search (LS) algorithm augmented with Deep Reinforcement Learning. We apply DeepLS for solving two illustrative NP-Hard routing problems in communication networks, namely, the OSPF Weight Setting problem (OWS) and the Routing and Wavelength Assignment problem (RWA). We chose OWS and RWA since they are both well-studied problems in communications literature for which strong algorithmic baselines are available, and are often solved as subproblems in high-level network

Nicola Di Cicco, Mamedhe Ibrahim, Sebastian Troia, and Massimo Tornatore are with the Department of Electronics, Information and Bioengineering (DEIB), Politecnico Di Milano, Italy. E-mail: {name}.{surname}@polimi.it

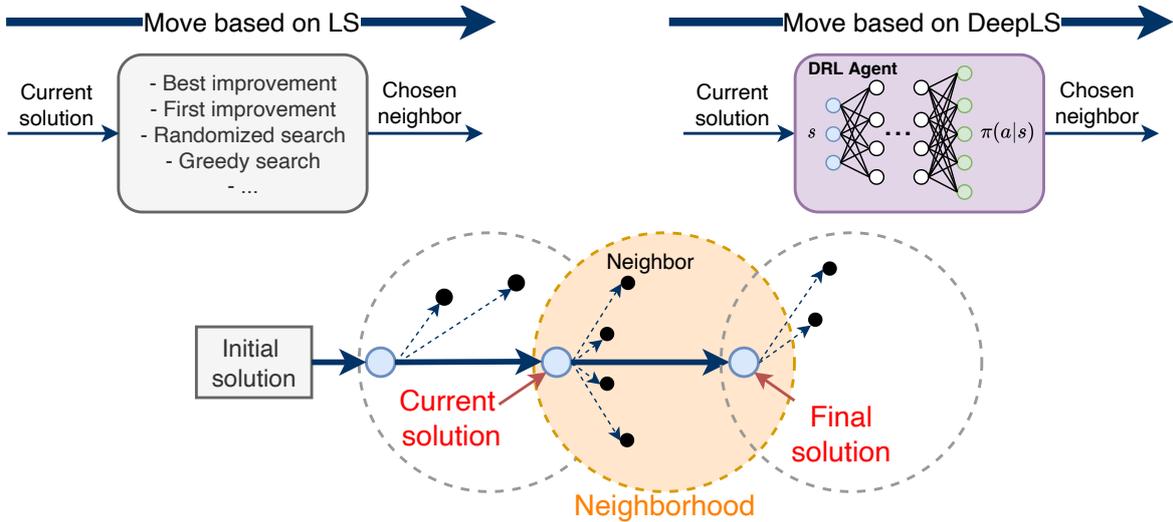


Fig. 1. Local Search (LS) vs. Deep Reinforcement Learning-augmented Local Search (DeepLS). Given an initial solution, traditional Local Search algorithms iteratively explore the search space following deterministic policies. However, designing the best neighbourhood choice policy for a given problem family is not straightforward. As such, the choice of a neighbour can be offloaded to a lightweight DRL agent, with the goal of learning a smart neighbourhood selection policy tailored for the specific optimization problem that is being solved.

management frameworks.

DeepLS leverages a DRL agent for guiding the LS neighbor selection procedure, with the goal of achieving better solutions in the long-term. Specifically, in Fig. 1 we illustrate how DRL can be used to augment a standard LS procedure: instead of applying myopic decision criteria, a DRL agent is trained to select the sequence of neighbours leading to the best final solution. To this end, we employ a lightweight permutation-equivariant artificial neural network architecture, composed of less than a hundred parameters, that can perform inference for problem instances of arbitrary size. As DRL agents often require a large number of environment interactions for learning an effective policy, we devise two fundamental countermeasures for shortening the required training times: i) we train our DRL agent only on one small network topology, which is significantly less complex to simulate, and ii) we train our DRL agent only for a small number of LS iterations, where it can at best converge to a local minima close to the starting solution. This allows to dramatically reduce training times down to few minutes, compared to the tens of hours required for state-of-the-art DRL-based approaches proposed in previous literature. In terms of solution quality, our approach outperforms both state-of-the-art DRL-based approaches and is competitive with handcrafted metaheuristics. To summarize, our results show that a simple Local Search heuristic, if augmented with lightweight DRL, can compete with far more elaborated handcrafted algorithms requiring minimal training effort and inference times overhead. These insights open up exciting research directions in many applications of Machine Learning for network optimization.

The remainder of this paper is organized as follows. In Section II we outline related works on RL applied to optimization problems in communications networks. In Section III we provide essential background notions on Reinforcement Learning and Deep Reinforcement Learning. In Section IV

we outline our proposed Local Search methodology, framing it into a Markov Decision Process. We illustrate the design of our lightweight neural architecture, and we outline its application to the OSPF Weight Setting (OWS) problem and the Routing and Wavelength Assignment (RWA) problem in optical networks. In Section V we illustrate our experimental results on the OWS and RWA problems. In Section VI we outline our conclusions and future research directions.

II. RELATED WORK

In this Section we briefly survey recent works in the field of RL applied to optimization in communications network. Moreover, we outline recent progress on the application of RL for enhancing Local Search algorithms.

A. Reinforcement Learning for Network Optimization

RL applied to optimization and control of communication networks has received significant attention in the recent years. Since RL is a universal methodology for data-driven sequential decision making, many applications have been explored in the networking literature. In the following, we overview recent studies on RL applied to communication networks.

In [3], a DRL agent based on a Graph Neural Network (GNN) is used for learning a heuristic for the Multi-Commodity Flow (MCF) problem. The DRL agent learns the OSPF weights optimizing the min-max link load in the network. The proposed approach attains competitive results to a state-of-the-art optimizer [13] in significantly shorter computing times. The DRL agent can be applied to graphs of any size, generalizing to topologies not seen during training.

In [14], a DRL agent is used to learn a heuristic for the MCF problem. The DRL agents learn edge weights that are then converted to per-flow splitting ratios via ‘‘SoftMin routing’’. The work in [15] extends [14] with the use of GNNs. In particular, authors emphasize on the capability of GNNs

to perform inference and generalize on graphs of arbitrary size. Both frameworks require solving the associated Linear Program of the MCF problem at every learning iteration, hence optimal solutions may not be attainable for larger problem instances or for NP-Hard optimization problems.

In [5], DRL is used to learn a heuristic algorithm for dynamic Routing, Modulation and Spectrum Assignment (RMSA) in optical networks. Given a connection request, the DRL agent either selects one among the K pre-computed paths and a feasible spectrum allocation or issues a proactive rejection. The proposed approach outperforms baseline greedy heuristics such as Shortest-Path First-Fit and K-Shortest-Paths First-Fit, but no comparison is provided against more competitive metaheuristics.

In [6], DRL is used to learn a heuristic algorithm for static Routing and Wavelength Assignment (RWA). The DRL agents operates similarly as in [5], but countermeasures are taken for large problem instances for which the sparsity of the reward hindered the final performance. The proposed approach outperforms baseline greedy heuristics and is competitive with a state-of-the-art Genetic Algorithm tailored for RWA, with significant savings in computing times.

In [16], DRL is used to learn a slice admission control policy in a 5G Radio Access Network (RAN). The objective is to maximize the revenue of the operator considering low and high priority slices, balancing penalties coming from rejecting a slice or failing to scale an already admitted slice due to resource unavailability. The proposed approach outperforms both static and threshold-based heuristics.

B. Reinforcement Learning alongside Local Search

Recently, several works have investigated the use of RL alongside Local Search for solving NP-Hard optimization problems. Indeed, applying RL to further improve general-purpose and well-performing heuristics is gathering attention in the research community.

In [17], DRL is used alongside Simulated Annealing for solving a maintenance planning problem. DRL is used to learn a Local Search algorithm that iteratively refines the best solution found by each iteration of SA. The proposed approach outperforms baseline SA and other data-driven heuristics.

In [7], DRL is used alongside Simulated Annealing for solving classical combinatorial optimization problem, such as the Knapsack Problem, the Bin Packing Problem and the Travelling Salesman Problem. DRL is used to learn the proposal function that is internally called by SA at each iteration. The proposed approach, albeit being a general-purpose solution method, attains competitive results to specialized heuristics for the considered problems.

In [18], a DRL-based Iterated Local Search was developed for solving the additive manufacturing machine scheduling problem. In particular, the Local Search procedure was implemented as a Variable Neighbourhood Search algorithm, for which the DRL agent selects the best neighbourhood to explore at each time-step. The proposed approach outperforms an evolutionary algorithm on medium to large problem instances.

C. Contributions

In the context of ML for Network Optimization, previous literature mostly focused on end-to-end learning [2], [5], [6], [14]. Compared to these works, we focus on how to augment an existing optimization method, e.g., a Local Search algorithm, with lightweight DRL-based intelligence.

In the context of DRL applied to Local Search algorithms, differently from [17] and [18], the DRL agent in our proposed solution learns *to choose the best move in a neighbourhood* rather than *to choose a specific neighbourhood structure*. As in [7], we use a Deep Set-like architecture [19] for implementing our neural network, but we did not opt for an augmented SA. From preliminary results, we have found the probabilistic rejections of SA to be detrimental for learning. We speculate that with simple LS, the DRL agent can learn by itself to balance between exploration and exploitation, and that the probabilistic rejections of SA may inadvertently pollute exploration. Moreover, since the Markov chains traversed by SA are notoriously very long and networking environments may be cumbersome to simulate, we concluded that Simulated Annealing as a methodology is not compatible with our requirements of short computational times.

In our work, we improve over the state-of-the-art by dealing with several research challenges of practical importance for a successful integration of ML in network optimization:

- **Training times:** DRL is notorious for being extremely sample inefficient, often requiring hours-long training before converging to a reasonable policy. DeepLS exploits a smart training procedure that allows to reach convergence in few minutes of training.
- **Generalization capabilities:** generally speaking, algorithms should work well for a broad and diversified set of problem instances. DeepLS shows remarkable generalization capabilities: while being trained on only one small network topology, it shows remarkable generalization capabilities to larger network topologies and traffic matrices unseen at training time.
- **Computational times:** Integrating Deep Neural Networks inside an optimization algorithm might introduce a significant overhead in the overall computational time. DeepLS leverages a lightweight neural network of less than a hundred parameters, therefore introducing negligible additional complexity in the Local Search procedure. Depending on the complexity of the optimization problem, computational times range from few seconds to few tens of seconds, which are fully in-line with what it is expected from state-of-the-art metaheuristics.
- **Solution quality:** DeepLS outputs solutions that outperform previous state-of-the-art DRL-based algorithms and either match or outperform significantly more complex handcrafted metaheuristics. In particular, the most relevant related work to ours is [3], in which authors propose an iterative DRL-based procedure for setting OSPF weights that leverages Graph Neural Networks (GNNs). Compared to [3], we achieve better performance with significantly shorter training/inference times, and without the need for a complex GNN model.

III. BACKGROUND

In this section we provide essential theoretical background on RL and DRL. Moreover, we outline the main definitions and properties of permutation-equivariant neural networks.

A. Reinforcement Learning

RL is a subset of ML which targets sequential decision-making problems. An agent interacts with a dynamic environment by taking actions, with the goal of maximizing the accumulation of rewards over a time horizon [20]. Formally, an RL environment is often modelled as a Markov Decision Process (MDP). An MDP is represented by the tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle$, where \mathcal{S} is the set of states, \mathcal{A} is the set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the state transition probability matrix, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, and $\mu : \mathcal{S} \rightarrow [0, 1]$ is the set of initial probabilities. We define the discounted return at time step t as $G_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k$.

The goal of an RL agent is to learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected discounted accumulation of rewards over a (possibly infinite) time-horizon T , as follows:

$$\arg \max_{\pi} J(\pi) = \mathbb{E} \left[\sum_{t=0}^T \gamma^t R_t \mid \pi \right] \quad (1)$$

Given a policy π , we define its value function $V_{\pi} : \mathcal{S} \rightarrow \mathbb{R}$ as $V_{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$, and its action-state value function $Q_{\pi} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ as $Q(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$.

In the following, we will consider episodic RL settings in which the time horizon T is assumed to be finite. For example, in RL applied to optimization, we can consider an episode to be corresponding to a single problem instance. In such episodic settings, the discount factor γ in Eq. (1) is often dropped, leading to the optimization of the undiscounted sum of rewards over an episode [21]. The shape of the reward function is a critical design choice in RL, as it ultimately defines the problem we are solving. Interestingly, whether or not “reward is enough” to achieve every conceivable optimization goal is currently an open research question [22].

In many applications of practical interest, the number of states and actions in an MDP may either be combinatorial or infinite. For instance, decision variables in an Integer Linear Programming (ILP) optimization problem may take a combinatorial number of possible values. Because of this it may not be possible, due to time and memory constraints, to represent and solve the MDP via exact methods. As such, we need to leverage a function approximation for solving complex MDPs via RL. In particular, the seminal paper of Mnih et al. [23] illustrated that it is possible to learn to play Atari games via Reinforcement Learning by using deep convolutional neural networks as function approximators, giving rise to the field of DRL. Since then, significant results have been achieved in a plethora of complex tasks using DRL, from continuous control to playing modern real-time strategy games [24]–[26].

B. Permutation-equivariant neural networks

A generic algorithm for network optimization operates on sets of decision variables. As such, to apply DRL to network

Algorithm 1 DRL-Augmented Local Search (DeepLS)

Require: Objective function $O(\cdot)$, starting solution s_0 (e.g., via a greedy heuristic), neighbour selection policy $\pi(a|s)$, transition function $\text{Step}(\cdot)$

```

1:  $s_{\text{best}} \leftarrow s_0$ 
2:  $O_{\text{best}} \leftarrow O(s_0)$ 
3: for  $t \leftarrow 0$  to  $N_{\text{iter}} - 1$  do
4:    $a_t \leftarrow \pi(a|s = s_t)$ 
5:    $s_{t+1} \leftarrow \text{Step}(s_t, a_t)$ 
6:   if  $O(s_{t+1}) < O_{\text{best}}$  then
7:      $s_{\text{best}} \leftarrow s_{t+1}$ 
8:   end if
9: end for
10: return  $s_{\text{best}}$ 

```

optimization, we need to leverage neural network architectures that can satisfy two fundamental properties. First, we want a neural network able to perform inference from an arbitrary number of inputs. This is because we want to apply a trained model on problem instances different (and possibly larger) than the ones seen during training. Second, we want the outputs of the neural network to be equivariant with respect to the ordering of its inputs. Indeed, even if we permute the ordering (i.e., the indexing) of the decision variables in an optimization problem, the problem that is being represented remains the same. Formally, a multi-output neural network is permutation-equivariant if the following holds:

$$\mathbf{f}_{\theta}(\text{perm}(\mathbf{x})) = \text{perm}(\mathbf{f}_{\theta}(\mathbf{x})) \quad (2)$$

where $\mathbf{f}_{\theta}(\cdot)$ is the neural network parameterized by θ , \mathbf{x} is the vector of inputs and $\text{perm}(\cdot)$ is a permutation operator. In practice, if we apply a permutation on the inputs, the same permutation is reflected in the outputs. For example, in the context of optimization problems, each input may correspond to a decision variable, and outputs may correspond to scores for each variable. As such, for the scores to be consistent, the neural network needs to be permutation equivariant.

In [19] authors outline general principles for designing permutation equivariant neural network layers. A permutation-equivariant layer can be generally expressed as follows:

$$\mathbf{f}(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{\Lambda} + \mathbf{1}\mathbf{1}^{\top}\mathbf{x}\mathbf{\Gamma}) \quad (3)$$

where $\mathbf{\Lambda}, \mathbf{\Gamma} \in \mathbb{R}^{m \times l}$ are learnable parameters, $\mathbf{x} \in \mathbb{R}^{n \times m}$ are the inputs, and $\sigma(\cdot)$ is an element-wise non-linearity. In particular, the input matrix \mathbf{x} represents n elements characterized by m features each. The transformations $\mathbf{x}\mathbf{\Lambda}$ and $\mathbf{x}\mathbf{\Gamma}$ are permutation equivariant, since they are equivalent to applying the linear transformation $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$, respectively, to each row of \mathbf{x} . Therefore, if a permutation is applied to the input elements \mathbf{x} , the same permutation is reflected on the elements of $\mathbf{f}(\mathbf{x})$. Stacking multiple layers defined as Eq. (3) allows to build permutation-equivariant deep neural networks.

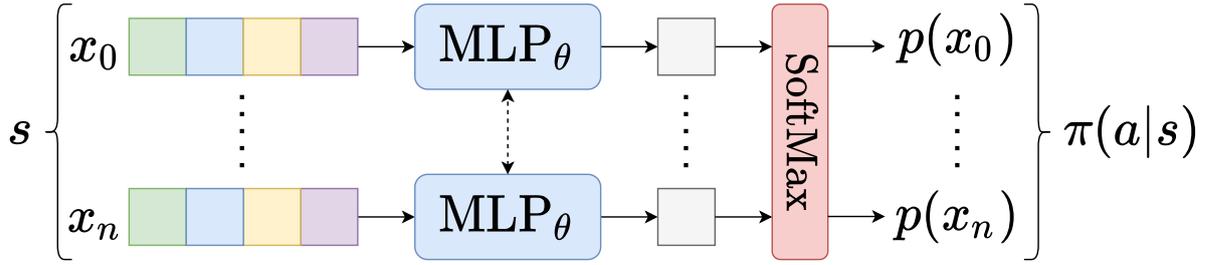


Fig. 2. DeepLS policy network architecture for a discrete action space. An action consists in choosing one among the n decision variables. Each decision variable x_0, \dots, x_n is associated with a set of features (e.g., if we associate one decision variable to a link in a network topology, link features can be the current link load, the current link weight, and so on). Features are processed in parallel by the same single-output neural network. Individual outputs are then aggregated (e.g., via SoftMax) to produce the policy $\pi(a|s)$.

IV. DEEPLS: DRL-AUGMENTED LOCAL SEARCH

Our proposed DRL-Augmented Local Search algorithm (DeepLS) is outlined in Algorithm 1. At each execution step, the neighbour selection policy implemented by a DRL agent $\pi(a|s)$, chooses a move by observing only the current solution. The Step function updates the current solution according to the move chosen by the policy. After N_{iter} iterations, the best solution found is returned. The transitions from a solution to the next one depend only on the current solution, therefore this Local Search procedure realizes a Markov chain over the solution space. We extend this Markov chain to an episodic MDP, which we aim to solve via model-free RL. In particular, in the following we will refer to a full execution of Algorithm 1 for some problem instance as an episode.

In the remainder of this Section, we outline our lightweight neural network architecture for implementing the DRL agent $\pi(a|s)$, and we illustrate its application to the OWS and RWA problems.

A. Neural network architecture

An ML-based solution method for solving optimization problem should have the capability to *generalize* to problem instances unseen during training [11]. In the context of network optimization, we are particularly interested in neural architectures enabling inference to network topologies of arbitrary size, possibly larger than the ones seen during training. Previous works in [3], [15] propose algorithms based on Graph Neural Networks (GNNs) for solving traffic engineering problems. Though we share the intuition behind the use of GNNs in the context of communication networks, in this work we propose a simpler alternative based solely on Multi-Layer Perceptrons (MLPs). In particular, as illustrated in [19] and suggested in [7], we employ an MLP-based architecture that can perform inference to instances of arbitrary size and is equivariant with respect to the ordering of the inputs. Our architecture sidesteps the complexity of the iterative message-passing procedure of GNNs¹.

¹In GNNs, to ensure that each node receives information from every other node, one needs a number of message-passing iterations equal to the network diameter. In sparse graphs (e.g., an optical network), GNNs with several hidden layers may suffer from issues such as oversmoothing [27] and oversquashing [28]. Moreover, large inference times in deep GNNs are among the most critical research challenges in the graph learning community [29].

We consider a neural architecture operating on sets of decision variables (e.g., the weights associated to each link in a network topology). Let n be the number of decision variables in our problem instance. We associate m features to each decision variable, hence, a state s is a matrix $s \in R^{n \times m}$. Let MLP_θ be an MLP neural network parameterized by θ with m inputs and one output. We can therefore process in parallel, via batching, all decision variables' features via the same single-output neural network MLP_θ . With reference to Eq. (3), our architecture is equivalent to stacking multiple layers of the form $\mathbf{o}^{l+1} = \sigma(\mathbf{o}^l \Lambda_l)$, where \mathbf{o}^l and Λ_l are the parameters and the outputs of the l -th MLP layer, respectively. We did not find the additional transformation Θ to benefit performance, and we opted for keeping the number of parameters to the minimum. The policy $\pi(a|s)$ can be therefore derived by aggregating the outputs of each forward pass in MLP_θ , e.g., via SoftMax.

A schematic representation of a policy network for a discrete action space, i.e., where the policy $\pi(a|s)$ defines a categorical distribution, is illustrated in Fig. 2. In particular, an action corresponds to choosing one among the decision variables. Outputs from each forward pass of MLP_θ are considered as non-normalized logits, which are then transformed via the SoftMax operator. The outputs of the SoftMax define a categorical distribution over the decision variables, realizing the discrete action space policy $\pi(a|s)$. Since each decision variable shares the same MLP_θ , the realized policy $\pi(a|s)$, i.e., the SoftMax aggregation over each individual output from MLP_θ , is permutation-equivariant with respect to the input decision variables. This fundamental property makes it possible to support arbitrarily large state and action spaces, which allows to apply the same pre-trained policy to problem instances of arbitrary size without the need for retraining.

Actor-Critic DRL algorithms (e.g., PPO [25]) require both an ‘‘Actor’’ network, parameterizing the policy $\pi(a|s)$, and a ‘‘Critic’’ network, parameterizing the value function $V(s)$. We can use the same architecture of Fig. 2 for implementing the Critic network. For a discrete action space, we consider each single output of MLP_θ to represent an action-value function $Q_\phi(s, a)$. From the action-value function, from which we can compute the value function $V_\phi(s) = \mathbb{E}_{\pi_\theta(a|s)}[Q_\phi(s, a)]$.

Thanks to fast batch parallelization, this neural architecture can be very efficiently applied to problem instances of arbitrary

size. Moreover, as outlined in Section III-B, this architecture is permutation equivariant with respect to the ordering of its inputs, i.e., the decision variables. We highlight that our architecture resembles the ‘‘Readout’’ module placed before the final outputs of a GNN [30].

At evaluation time we aim to assess the generalization capabilities of DeepLS to problem instances unseen during training. In particular, with our experimental results, we quantitatively answer the following research questions:

- 1) Can our approach generalize to traffic distributions unseen during training?
- 2) Can our approach generalize to larger network topologies unseen during training?
- 3) Can our approach, if trained on short episodes (i.e., executions of Algorithm 1 where N_{iter} is small) generalize to longer episodes, therefore achieving better solutions in the long run?

B. DeepLS for OSPF Weight Setting

1) *Premise on the Multi-Commodity Flow problem (MCF)*: optimal traffic engineering can be achieved by solving the well-known Multi-Commodity Flow (MCF) problem. The MCF problem can be formulated as a Linear Programming (LP) optimization problem, hence it can be solved in polynomial time. Let $G(V, E)$ be a bidirectional graph where each edge $(u, v) \in E$ has capacity $c_{u,v}$. Let $d_{s,t}$ be the flow units that need to be routed from the source node $s \in V$ to the destination node $t \in V$. We define continuous decision variables $f_{u,v}^t$, representing the amount of flow with destination $t \in V$ routed on edge $(u, v) \in E$. As objective function, we consider the minimization of the maximum normalized link load. The LP formulation of the MCF problem reads as follows:

$$\min \max_{(u,v) \in E} \sum_{t \in V} \frac{f_{u,v}^t}{c_{u,v}} \quad (4)$$

$$\sum_{t \in V} f_{u,v}^t \leq c_{u,v} \quad \forall (u, v) \in E \quad (5)$$

$$\sum_{v \in V} f_{s,v}^t - \sum_{u \in V} f_{u,s}^t = d_{s,t} \quad \forall s, t \in V, s \neq t \quad (6)$$

$$f_{u,v}^t \geq 0 \quad \forall t \in V, \forall (u, v) \in E \quad (7)$$

Objective function (4) minimizes the maximum normalized link load, Eq. (5) are the maximum link capacity constraints, and Eq. (6) are the flow conservation constraints. Being this problem an LP, its optimal solution can be computed very efficiently by commercial solvers.

2) *The OSPF Weight Setting problem (OWS)*: let us now consider the case in which routing configurations are determined by link-state protocols, such as OSPF, in which routes are computed using Dijkstra’s algorithm on the weighted network topology graph. In this context, the problem of determining the edge weights realizing the optimal routing of (4)-(7) is NP-Hard [31]. We therefore consider the problem of finding the OSPF weights that minimize the maximum link load in the network (4). We also consider Equal Cost Multi-Path (ECMP) routing, i.e., flows directed to the same destination are equally distributed among all shortest paths of equal cost.

3) *Formulating LS for OWS as an MDP*: we consider an LS algorithm where, at each iteration, the weight of one link is increased and a new OSPF routing is computed.

State Space: we consider a state space where each link in the network is associated to two features: *i)* normalized link load, and *ii)* link weight. This is the same set of features that was used in [3], for a fair comparison between the two approaches.

Action Space: we consider a discrete action space of dimension $|E|$. An action consists in choosing a link. The chosen link has its associated weight incremented by one unit. Integer weights allow for the possibility of having multiple paths with the same total weight, in which flows will be equally split as per ECMP. Again, to keep the comparisons fair, this is the same action space that was used in [3].

Reward Function: we consider the immediate gain $R_t = O_{t+1} - O_t$, where O_t is the maximum normalized link load at iteration t . We also experimented with the absolute improvement $R_t = \max\{0, O_t^{\text{best}} - O_t\}$, where O_t^{best} is the current best maximum link load found up to iteration t , but we observed little difference in performance.

Worst-case Complexity: each iteration of DeepLS requires recomputing the all-pairs shortest paths after the link weight update and the new OSPF routing. In the worst case, computing the all-pairs shortest paths in sparse graphs is $O(|V|^2 \log(|V|))$ with Johnson’s algorithm. The forward pass in the DeepLS MLP neural network, considering it as a naive matrix multiplication, is $O(|E|F)$, where F is the number of edge features, and assuming the number of hidden neurons and hidden layers to be constant terms. Finally, computing the new OSPF routing with full traffic matrix is $O(|V|^3)$, since the number of hops in the routes are bounded by the total number of nodes. Assuming $F \ll |V|$, the worst-case time complexity of a single DeepLS iteration for OWS is $O(|V|^2 \log(|V|) + |V|^3) = O(|V|^3)$. The overall worst-case complexity for running DeepLS for N_{iter} iterations is therefore $O(N_{\text{iter}}|V|^3)$.

C. DeepLS for Routing and Wavelength Assignment

1) *The Routing and Wavelength Assignment problem (RWA)*: RWA consists in assigning a route and a wavelength to a set of connection requests in an optical Wavelength Division Multiplexing (WDM) network. We assume transparent optical switching, hence wavelength continuity constraints must be enforced for each established lightpath. The objective is to maximize the number of established lightpaths. RWA can be expressed as an Integer Linear Programming (ILP) optimization problem as follows:

$$\max \sum_{p \in P} \sum_{w \in W} x_p^w \quad (8)$$

$$\sum_{p|l \in p} x_p^w \leq 1 \quad \forall l \in E, \forall w \in W \quad (9)$$

$$\sum_{p \in P(s,d)} \sum_{w \in W} x_p^w \leq \rho(s,d) \quad \forall s, d \in V, s \neq d \quad (10)$$

$$x_p^w \in \{0, 1\} \quad \forall p \in P, \forall w \in W \quad (11)$$

Where x_p^w indicates whether or not wavelength w is occupied in path p , P is a set of pre-computed paths, and $\rho_{(s,d)}$ is the number of connection requests between nodes s and d . Constraints (9) are wavelength continuity constraints, whereas constraints (10) ensure that the maximum number of wavelengths per link is not exceeded.

2) *Formulating an LS for RWA as an MDP*: we consider an LS algorithm where, at each iteration, the weight of one link is increased and a new feasible solution is computed with K-Shortest-Paths First-Fit (KSP-FF) on the weighted graph.

State Space: we consider a state space where each link in the network is associated to three features: *i*) normalized link load, *ii*) link weight, and *iii*) link betweenness centrality, defined as the fraction of shortest paths that traverse that edge. We empirically assessed that introducing this additional graph-level feature leads to improved performance.

Action Space: we consider a discrete action space of dimension $|E|$. An action consists in choosing a link. The chosen link has its associated weight incremented by one unit.

Reward Function: we consider the immediate gain $R_t = O_{t+1} - O_t$, where O_t is the blocking rate at iteration t .

Worst-case Complexity: each iteration of DeepLS requires computing the all-pairs K-shortest simple paths after the link weight update² and the routing and wavelength assignment with KSP-FF. Computing the K-shortest simple paths for a node pair in sparse graphs is $O(K|V|^2 \log(|V|))$ with Yen’s algorithm. Therefore, computing the all-pairs K-shortest simple paths by iteratively calling Yen’s algorithm is $O(K|V|^4 \log(|V|))$. As before, the forward pass in the DeepLS MLP neural network, being a matrix multiplication, is $O(|E|F)$. Finally, to evaluate the new solution value, we need to perform routing via KSP-FF. Determining the availability of a continuous wavelength over a path is $O(|V|W)$, where W is the number of wavelengths per link. Following up, checking for wavelength availability in each of the K-paths is $O(K|V|W)$. Routing all demands is therefore $O(K|V|WD)$, where D is the total number of demands. Assuming $F \ll |V|$, the worst-case time complexity of a single DeepLS iteration for RWA is $O(K|V|^4 \log(|V|) + K|V|WD)$. The overall worst-case complexity for running DeepLS for N_{iter} iterations is therefore $O(N_{\text{iter}}K(|V|^4 \log(|V|) + |V|WD))$

V. ILLUSTRATIVE EXPERIMENTAL RESULTS

In our experiments, we used Proximal Policy Optimization (PPO) [25] for training the DRL agents, with the Stable Baselines implementation [32]. The Actor and Critic networks of PPO are implemented as the architecture illustrated in Fig. 2, where each MLP network has one single hidden layer with 16 neurons and Exponential Linear Unit (ELU) activation. In total, the two networks have $2 \cdot (m \cdot 16 + 16 + 16 + 1)$ trainable parameters, where m is the number of features associated to each decision variable. At inference time, only the policy network needs to be stored. In our problems, the number of parameters of the policy networks is 65 and 81 for

²As the link weights can only be increased, path recomputation can be performed only for the node pairs for which at least one of the K-shortest paths traverses the modified link. While not improving the worst-case asymptotic complexity, this leads to much shorter computational times in practice.

OWS and RWA, respectively. Moreover, in all experiments we train on short episodes and evaluate on episodes at least one order of magnitude longer than training. Finally, being the environments CPU-bound, we leverage multiprocessing to run 6 environments in parallel for gathering experience. All of these tweaks allow to significantly reduce training times down to few minutes without sacrificing performance. All experiments were ran on a workstation with an Intel Core i5-8400 CPU (6 cores @2.80 GHz) and 32 GB RAM.

Our source code is publicly available at the following link: <https://github.com/bonsai-lab-polimi/tnsm2023-deepls>

A. OSPF Weight Setting problem

Training: we train only on the NSFNet topology with uniform traffic distribution. For fair comparison, we consider the same 100 training traffic matrices that were used in [3]. We train our model for 30000 episodes of length equal to $N_{\text{iter}}=10$. In each episode, a traffic matrix is randomly sampled from the 100 training ones. For PPO, we set the rollout buffer size to 128, and all other hyperparameters are left as their default values in the Stable-Baselines3 [32] implementation. The starting solution for DeepLS is computed as a standard OSPF routing with all links having unit weight. This means that our agent will be able to explore a small neighbourhood in which the total difference between the starting weights and the final weights can be at most equal to 10. On our hardware, training takes approximately 6 minutes. This is an improvement of more than two orders of magnitude compared to MARL-GNN, whose training times elapsed on average 11 hours on our hardware.

Evaluation: we evaluate on NSFNet, GBN and GEANT2 with both uniform and gravity-based traffic distributions, considering 100 test traffic matrices for each network-traffic combination. The traffic matrices used for testing are the same that were used in [3]. We therefore evaluate the generalization capabilities of DeepLS both on traffic distributions and network topologies unseen during training. We evaluate DeepLS on episodes of length $N_{\text{iter}}=100$ steps for NSFNet, $N_{\text{iter}}=150$ for GBN and $N_{\text{iter}}=200$ for GEANT2, for fair comparison to [3]. Results are averaged over five different training seeds.

Baselines: we compare DeepLS against the following:

- **Default OSPF**: OSPF routing resulting from having all link weights equal to one unit.
- **MARL-GNN** [3]: state-of-the-art GNN-based approach. We trained and evaluated MARL-GNN using the open-source code provided by the authors. For both training and inference we employed the same training configuration illustrated in the original paper. At inference time, we evaluate MARL-GNN in the same settings (i.e., topology and traffic distributions) in which it has been trained on. Results are averaged over five different training seeds.
- **LS-Greedy**: greedy local search heuristic which, at each timestep, selects the link that currently has the maximum link-load and increments its weight by one unit. For a fair comparison, LS-Greedy is run for a number of iteration $N_{\text{iter}}=100$, the same as DeepLS. The purpose of this baseline is to outrule the possibility that DeepLS learned a myopic greedy decision rule.

- **DEFO** [13]: state-of-the-art optimizer for traffic engineering, producing high-quality solutions. In particular, DEFO does not compute OSPF weights, but per-flow spitting ratios. These same solutions were considered as baseline in [3], which we extracted from their publicly available code repository. DEFO’s bytecode is publicly available at [33], and the solutions reported are the best found in the fixed 30s time limit.
- **LP**: for each traffic matrix we compute the optimal solutions for the LP (4)-(7), which we report as a lower bound for the best possible solution that can be achieved.

Numerical Results: in Fig. 3 we show the CDF of the maximum link load achieved by DeepLS and by the baselines on the test traffic matrices for NSFNet, GBNet and GEANT2. Generally speaking, we observe that DeepLS is able to generalize out-of-the-box to larger test set instances unseen at training time, outperforming MARL-GNN and attaining similar performance to DEFO in significantly shorter computational times. In particular, we underline that only DeepLS is evaluated on different network topologies and traffic distributions than training, whereas for MARL-GNN we kept the training and the evaluation settings the same.

Compared to LS-Greedy, DeepLS improves the maximum link load, on average, by 2.7% on NSFNet-Uniform, by 1.9% on NSFNet-Gravity, by 5.2% on GBNet-Uniform, by 0.8% on GBNet-Gravity, by 3.4% on GEANT2-Uniform, and by 3.0% on GEANT2-Gravity. As such, DeepLS consistently improves on the myopic LS-Greedy thanks to its lightweight policy.

Compared to the MARL-GNN [3], DeepLS improves the maximum link load, on average, by 5.6% on NSFNet-Uniform, by 1.7% on GBNet-Uniform, by 4.8% on GBNet-Gravity, by 2.6% on GEANT2-Uniform and by 1.5% on GEANT2-Gravity, while achieving slightly worse results (0.24% difference) only on NSFNet-Gravity. We remark that in NSFNet-Uniform, GEANT2-Uniform and GBNet-Gravity the GNN-based approach is also outperformed by the myopic LS-Greedy, whereas our DeepLS approach outperforms on average LS-Greedy in all of the considered scenarios³.

Furthermore, compared to the high-quality solutions computed by DEFO, DeepLS improves on average the maximum link load by 2.6% in GBNet-Gravity, by 9.6% in GEANT2-Uniform and by 1.4% in GEANT2-Gravity, while worsening it by 5.1% in NSFNet-Uniform, 2.4% in NSFNet-Gravity, and by 2.7% in GBNet-Gravity. Overall, DeepLS ties with DEFO, showing that it is on average capable to achieve solutions of similar quality than a state-of-the-art approach in highly diversified scenarios.

In terms of relative gap to the lower bound, DeepLS achieves on average a 10.2% relative gap, whereas the considered baselines achieve relative gaps equal to 10.6%, 13.3% and 14.24% for DEFO, LS-Greedy and MARL-GNN, respectively. Even though there is still a non-negligible optimality gap, on average DeepLS achieves slightly better performance

³This is somehow counterintuitive, as it appears that introducing a GNN module before the readout network is actually *worsening* the final performance. This can be imputed to a number of issues arising in GNNs such as oversmoothing [27] and oversquashing [28], which may hinder the overall final performance if not addressed properly.

than DEFO and significantly outperforms MARL-GNN. The optimality gap of DeepLS can be imputed to the constraint of equal flow split between shortest paths of equal cost. The LP can decide on arbitrary flow splits, hence providing only a lower bound on the optimal objective value for OWS.

We underline that OWS is a classical networking problem, extensively studied for several decades. We do not aim to outperform *all* state-of-the-art algorithm for OWS, but to demonstrate the potential of augmenting well-known algorithms with lightweight intelligence. In that regard, we have shown that a simple DRL-augmented Local Search becomes competitive with DEFO, a far more complex metaheuristic.

In Fig. 4 we illustrate the average running times per instance of the proposed DeepLS approach against LS-Greedy and the state-of-the-art MARL-GNN baseline. The times for LP are not shown, as the LP does not solve OWS, but provides only a theoretical lower bound on the solution value. We note that the times for DeepLS and LS-Greedy are nearly identical, showing that our lightweight neural architecture does not introduce additional overhead to the LS algorithm, while consistently improving on the solution quality. Compared to MARL-GNN, our lightweight MLP-only approach achieves on average an 83% speedup⁴, while achieving solutions of better quality and with training times two order of magnitude smaller. A detailed comparison between the worst-case time complexities of DeepLS for OWS and the considered baselines is provided in Appendix A-A.

B. Routing and Wavelength Assignment problem

Training: we train on the NSFNet topology assuming 10 wavelengths per link and traffic matrices of 100 uniformly-distributed source-destination requests. We train our model for 10000 episodes of length equal to $N_{\text{iter}}=10$. For PPO, we set the rollout buffer size to 128, and all other hyperparameters are left as their default values in the Stable-Baselines3 [32] implementation. For each episode, a traffic matrix is randomly generated from a uniform distribution between all possible source-destination pairs. The starting solution for DeepLS is computed via KSP-FF with all links having unit weight. On our hardware, DeepLS takes approximately 10.6 minutes to train.

Evaluation: we evaluate DeepLS on NSFNet, GEANT2 and a synthetic 50-node Gabriel graph. Gabriel graphs have been observed, among different graph generators, to better capture the structure of physical layer network topologies [34]. We consider networks with 80 wavelengths per link and 800 uniformly distributed source-destination requests. We evaluate on episodes of length equal to $N_{\text{iter}}=100$ for each network topology. Results are averaged over five training seeds.

Baselines: we compare DeepLS against the following:

- **KSP-FF:** default K-Shortest-Paths First-Fit routing with links having equal unit weight. We consider $K=3$ shortest paths for each source-destination pair.

⁴Authors of [3] used a slightly less time-efficient algorithm for recomputing the all-pairs shortest paths at each MARL-GNN iteration. With their algorithm, our relative speedup becomes 66% on average.

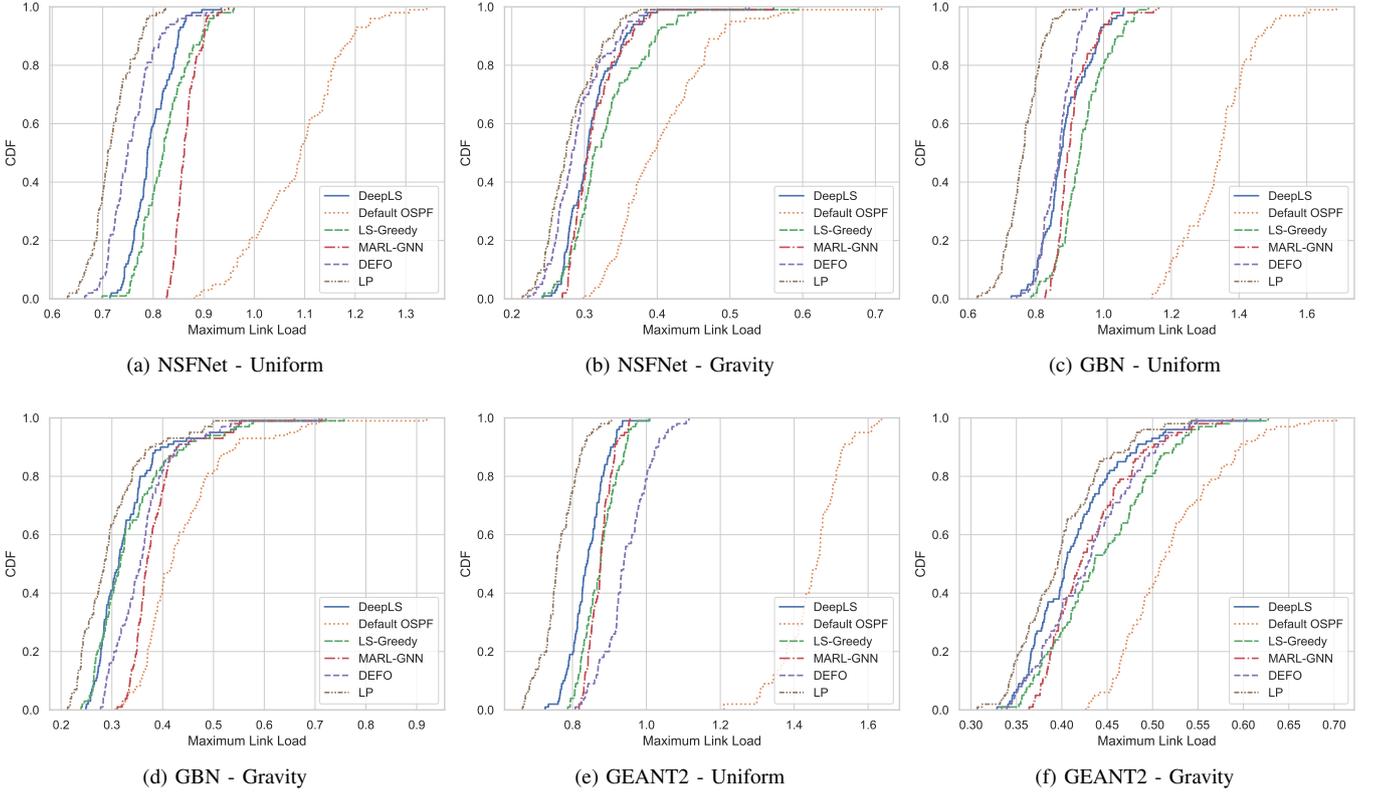


Fig. 3. CDFs of the maximum link load achieved by DeepLS and the considered baselines on 100 test traffic matrices.

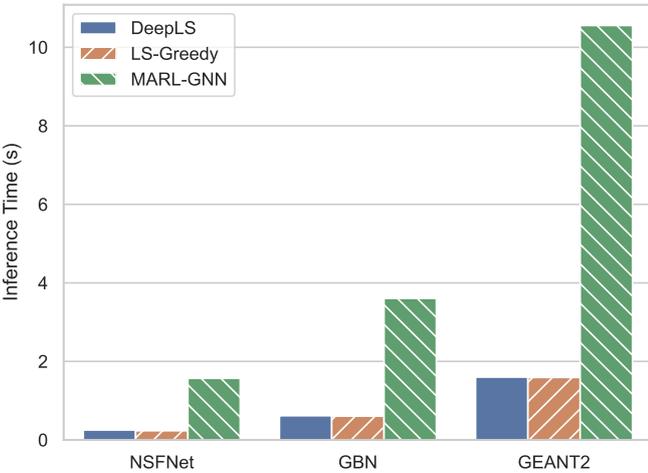


Fig. 4. Average computational times of DeepLS per OWS problem instance against LS-Greedy and the state-of-the-art MARL-GNN [3]. Computational times for DEFO [13] are fixed to 180s.

- **LS-Greedy**: greedy local search heuristic which, at each timestep, selects the link that currently has the maximum load and increments its weight by one unit. The link load is measured as the ratio between the number of occupied wavelengths and the number of available wavelengths. For a fair comparison, LS-Greedy is run for a number of iteration $N_{\text{iter}}=100$, the same as DeepLS.
- **PPO-FF** [6]: state-of-the-art DRL-based algorithm that

processes the traffic matrix sequentially while querying a DRL agent for routing and admission control. The DRL agent either chooses one among the available K -shortest paths or for issuing a proactive rejection. In case of request admission and routing, wavelength assignment is performed by first-fit.

- **GA-FF** [35]: state-of-the-art Genetic Algorithm for solving RWA. In particular, GA-FF optimizes only the routing assuming the precomputation of K -shortest paths, and performs wavelength assignment by first-fit. We consider $K=3$ shortest paths for each source-destination pair. We consider a large population size of 1000 individual, and we run the algorithm until it reaches convergence, i.e., the objective function value is not improving after a large number of iterations.
- **ILP**: for each traffic matrix we compute the optimal solutions of the ILP (8)-(11), which we report as a lower bound for the best possible solution that can be achieved. We consider $K=3$ shortest paths for each source-destination pair.

Numerical results: in Fig. 5 the CDFs of the blocking rates achieved by DeepLS and the considered baselines are shown. Similarly to OWS, we observed that DeepLS generalizes out-of-the-box for larger problem instances unseen during training, achieving from similar to better solutions than GA-FF in significantly shorter computational times.

Compared to LS-Greedy, DeepLS achieves a relative improvement of 17% on NSFNet, 32% on GEANT2 and 40% on the 50-node Gabriel graph. Remarkably, LS-Greedy com-

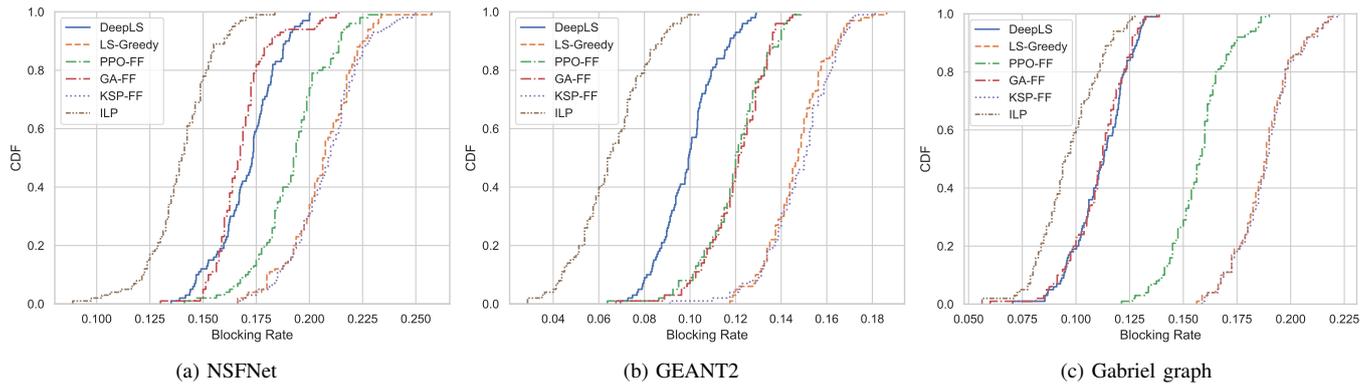


Fig. 5. Empirical CDFs of the blocking rates achieved by DeepLS and the considered baselines on 100 test traffic matrices.

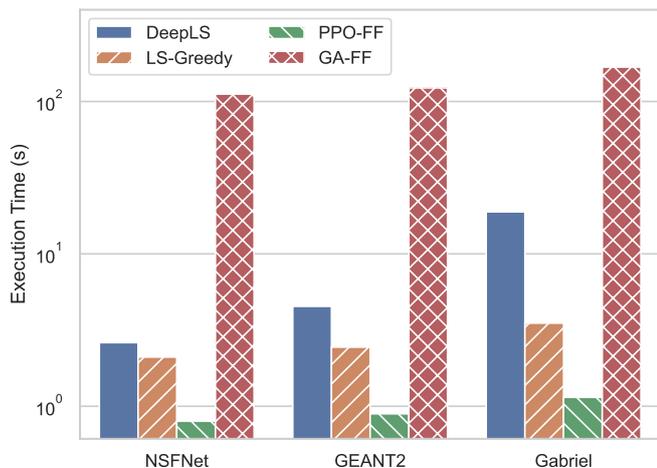


Fig. 6. Average computational times of DeepLS per RWA problem instance against a state-of-the-art Genetic Algorithm (GA) for RWA.

pletely fails to improve the starting KSP-FF solution. Empirically, we observed that LS-Greedy reaches stagnation almost immediately, repeatedly incrementing the weight of the same edge without improving the blocking rate. This illustrates that, unlike for the OWS problem, more sophisticated neighbourhood exploration strategies are required for achieving good quality solutions via Local Search.

Compared to PPO-FF, DeepLS achieves a relative improvement of 11% on NSFNet, 17% on GEANT2, and 29% on the 50-node Gabriel graph. These results illustrate that DeepLS brings a significant improvement in state-of-the-art DRL-based algorithms for solving static routing problems in optical networks. Moreover we underline that PPO-FF, while consistently outperforming KSP-FF, falls short against more complex traditional optimization algorithms such as GA-FF.

Finally, compared to GA-FF, DeepLS achieves solutions on average 17.6% better than GA-FF on GEANT2 and solutions of comparable quality on NSFNET and on the 50-node Gabriel graph (on average 1.6% and 0.8% worse, respectively). This is a remarkable result, showing that a weaker heuristic such as Local Search, enhanced by lightweight learning intelligence, can become competitive with respect to

a handcrafted metaheuristic.

In Fig. 6 we illustrate the computational times of DeepLS compared to LS-Greedy, PPO-FF and GA-FF. The times for ILP are not shown: while it is possible to solve RWA on NSFNET in few seconds, computational times for GEANT2 and the 50-node Gabriel graph are out of scale. Compared to LS-Greedy, DeepLS is 48% slower: we empirically verified that this is because DeepLS triggers more K-shortest paths recomputations compared to LS-Greedy, leading to a more effective exploration of the solution space. Indeed, as shown before, LS-Greedy fails to improve the starting KSP-FF solution. PPO-FF is the fastest among all baselines since, by design, its runtime is only slightly slower than standard greedy heuristics such as KSP-FF. Unfortunately, as previously outlined, it often falls short in terms of solution quality compared to advanced metaheuristics. Compared to GA-FF, DeepLS achieves on average a 93% speedup, with computational times equal at most to 20s for the 50-node Gabriel Graph. Overall, DeepLS can compute solutions either on-par or better than a fine-tuned GA in significantly shorter computing times. A detailed comparison between the worst-case time complexities of DeepLS and the considered baselines is provided in Appendix A-B.

Again, DeepLS is able to generalize to larger problem instances on network topologies unseen during training. This allows us to keep training times to the minimum, as it is possible to train only on the least computationally intensive settings (i.e., smaller network topologies, smaller network capacity and lower total number of demands).

In terms of optimality gap with respect to the ILP, DeepLS achieves on average a 21.9% gap, whereas the baselines achieve gaps equal to 45.4% and 25.1% for LS-Greedy and GA-FF, respectively. The non-negligible optimality gap of DeepLS can be imputed to the limitations of a First-Fit wavelength assignment strategy. We underline that, for DeepLS, the smallest optimality gap of 13.1% was observed for the largest problem instances, i.e., on the 50-node Gabriel graph.

Again, RWA is a well-known problem in networking. Our aim is not to beat all state-of-the-art approaches for RWA, but to illustrate that simple algorithms enhanced with lightweight artificial intelligence can become more powerful than significantly more complex handcrafted algorithms. In

our case, DeepLS either outperforming or matching GA on the largest instances in far shorter computing times highlights the disruptive potential of ML alongside OR.

VI. CONCLUSION

In this paper, we proposed a lightweight DRL-augmented Local Search algorithm (DeepLS), and we applied it to the OSPF Weight Setting problem and the Routing and Wavelength Assignment problem. Leveraging a permutation-equivariant neural network operating on sets of decision variables, DeepLS can evaluate problem instances of arbitrary size, showing remarkable generalization capabilities to network topologies, traffic distributions, network capacities and network loads unseen during training. DeepLS, composed by less than a hundred parameters, outperforms state-of-the-art DRL-based approaches in literature and achieves competitive performance compared to state-of-the-art metaheuristics, while being on average faster than both. Overall, our results show that incorporating lightweight learning intelligence in existing methodologies can be the cornerstone for designing powerful and scalable network-optimization algorithms. As the proposed methodology is general, future research directions include applying DeepLS to more complex neighbourhood definitions (e.g., rerouting, ejection chains), more complex network-optimization problems (e.g., Routing, Modulation format and Spectrum Assignment in Elastic Optical Networks, optimization of Service Function Chaining), and more sophisticated metaheuristic frameworks (e.g., Iterated Local Search, Variable Neighbourhood Search, Ruin&Recreate).

ACKNOWLEDGEMENT

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on ‘‘Telecommunications of the Future’’ (PE00000001 - program ‘‘RESTART’’).

APPENDIX A

WORST-CASE COMPLEXITY ANALYSIS

In this Appendix, we compare the worst-case complexities of the considered baselines against DeepLS for both the OSPF Weight Setting problem and the Routing and Wavelength Assignment problem.

A. OSPF Weight Setting problem

We previously demonstrated that the worst-case time complexity of DeepLS for OWS is $O(N_{\text{iter}}|V|^3)$. The worst-case time complexities of the considered baselines are as follows:

- **Default OSPF**: requires computing the all-pairs shortest paths and distributing the demands. Hence, as shown for a single iteration of DeepLS, the complexity is $O(V^3)$.
- **MARL-GNN** [3]: requires a forward pass on the line graph of the network topology. The time complexity of the forward pass is therefore proportional to the number of edges in the line graph of the network topology, i.e., $O(L|V||E|)$, where L is the number of GNN layers. As

the algorithm requires at each iteration to recompute all-pairs’ shortest paths and to distribute the demands, the overall time complexity is $O(N_{\text{iter}}(|V|^3 + L|V||E|))$.

- **LS-Greedy**: equal to DeepLS, as LS-Greedy misses only the neural network policy, whose computational complexity can be elided.
- **DEFO** [13]: with reference to [36], DEFO alternates between destruction and recreation procedures for generating an optimal forwarding plane configuration. The time complexity of recreation is $O(N_{\text{iter}}|E||V|^2 \log |V|)$, as it requires repeated sorting of the nodes and propagation of the link load constraints. The time complexity of destruction, compared to recreation, can be elided.

We underline that asymptotically running DeepLS is as complex as running the Default OSPF heuristic N_{iter} times. Indeed, the complexity is upper-bounded by the evaluation of the objective function value and the link loads. Finally, while MARL-GNN and DeepLS show similar worst-case time complexities, we illustrated in Fig. 4 that DeepLS scales much better with instance size thanks to its extremely lightweight neural network architecture.

B. Routing and Wavelength Assignment problem

We previously demonstrated that the the worst-case time complexity of DeepLS for RWA is $O(N_{\text{iter}}K(|V|^4 \log(|V|) + |V|WD))$. The worst-case time complexities of the considered baselines are as follows:

- **KSP-FF**: requires computing the all-pairs’ K-shortest paths and to distribute the demands. Therefore, the worst-case complexity is $O(K|V|^4 \log(|V|) + K|V|WD)$.
- **LS-Greedy**: equal to DeepLS, as LS-Greedy misses only the neural network policy, whose computational complexity can be elided.
- **PPO-FF** [6]: requires computing the all-pairs’ K-shortest paths and querying the DRL agent for each connection request in the traffic matrix. As the state space of PPO-FF is $O(|V|)$, the computational complexity of routing the full traffic matrix is $O(K|V|^2WD)$. Therefore, the computational complexity of PPO-FF is $O(K(|V|^4 \log(|V|) + |V|^2WD))$.
- **GA-FF** [35]: the algorithm optimizes the ordering of demands and the chosen routing, whereas wavelength assignment is performed with a first-fit rule. As such, the worst-case complexity of the algorithm is bounded by the fitness function calculation, i.e., the evaluation of the objective function value given a request ordering and a choice of routing for every request. Assuming that the all-pairs’ K-shortest paths have been precomputed, the overall time complexity is $O(N_{\text{iter}}K|V|WD \cdot \text{Pop_Size})$, where Pop_Size is the population size (i.e., the number of candidate solutions to be kept at each iteration of GA-FF). In practice, it holds that $\text{Pop_Size} = O(KD)$, as a common rule of thumb is to set the population size proportional to the number of variables (i.e., genes) in a candidate solution, which in our cases are one per demand. Hence, the overall time complexity is $O(N_{\text{iter}}K^2|V|WD^2)$.

Similarly to what we observed for OWS, running DeepLS is as time-consuming as running KSP-FF N_{iter} times. In this regard, while PPO-FF scales significantly better than DeepLS, as it does not have to recompute the K-shortest-paths, but significantly underperforms in terms of solution quality. Finally, we underline that while the time-complexities for DeepLS and GA-FF may be difficult to compare in the current form, it is reasonable to assume that $O(D) = O(|V|^2)$ (e.g., considering a full-mesh traffic matrix). Hence, the time-complexities of DeepLS and GA-FF further simplify to $O(KN_{\text{iter}}(|V|^4 \log(|V|) + W|V|^3))$ and $O(N_{\text{iter}}K^2W|V|^5)$, respectively. Therefore, DeepLS not only provides similar to better solutions than GA-FF, but also scales better with respect to the instance size, as we empirically demonstrated in Fig. 5 and Fig. 6.

REFERENCES

- [1] D. Andreoletti, S. Troia, F. Musumeci, S. Giordano, G. Maier, and M. Tornatore, "Network traffic prediction based on diffusion convolutional recurrent neural networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 246–251.
- [2] C. Natalino, C. Manso, L. Gifre, R. M. noz, R. Vilalta, M. Furdek, and P. Monti, "Microservice-based unsupervised anomaly detection loop for optical networks," in *Optical Fiber Communication Conference (OFC) 2022*. Optica Publishing Group, 2022, pp. 1–3.
- [3] G. Bernárdez, J. Suárez-Varela, A. López, B. Wu, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Is machine learning ready for traffic engineering optimization?" in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, 2021, pp. 1–11.
- [4] D. Aureli, A. Cianfrani, M. Listanti, and M. Polverini, "Intelligent link load control in a segment routing network via deep reinforcement learning," in *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2022, pp. 32–39.
- [5] X. Chen, J. Guo, Z. Zhu, R. Proietti, A. Castro, and S. J. B. Yoo, "Deep-rmsa: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks," in *2018 Optical Fiber Communications Conference and Exposition (OFC)*, 2018, pp. 1–3.
- [6] N. Di Cicco, E. F. Mercan, O. Karandin, O. Ayoub, S. Troia, F. Musumeci, and M. Tornatore, "On deep reinforcement learning for static routing and wavelength assignment," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 28, no. 4, pp. 1–12, 2022.
- [7] A. H. C. Correia, D. E. Worrall, and R. Bondesan, "Neural simulated annealing," 2022. [Online]. Available: <https://arxiv.org/abs/2203.02201>
- [8] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016.
- [9] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, 2015.
- [10] M. Gasse, D. Chetelat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, 2019.
- [11] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.
- [12] "EURO Meets NeurIPS 2022," <https://euro-neurips-vrp-2022.challenges.ortec.com/>, 2022, [Online; accessed 11-December-2022].
- [13] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, p. 15–28.
- [14] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, p. 185–191.
- [15] O. Hope and E. Yoneki, "Gddr: Gnn-based data-driven routing," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 517–527.
- [16] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti, "Reinforcement learning for slicing in a 5g flexible ran," *Journal of Lightwave Technology*, pp. 1–12, 06 2019.
- [17] F. Kosanoglu, M. Atmis, and H. Turan, "A deep reinforcement learning assisted simulated annealing algorithm for a maintenance planning problem," *Annals of Operations Research*, 03 2022.
- [18] M. Alicastro, D. Ferone, P. Festa, S. Fugaro, and T. Pastore, "A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems," *Computers & Operations Research*, vol. 131, 2021.
- [19] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.
- [21] C. Nota and P. S. Thomas, "Is the policy gradient a gradient?" 2019. [Online]. Available: <https://arxiv.org/abs/1906.07073>
- [22] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, vol. 299, p. 103535, 2021.
- [23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: <http://arxiv.org/abs/1312.5602>
- [24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: <http://arxiv.org/abs/1509.02971>
- [25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: <https://arxiv.org/abs/1707.06347>
- [26] O. Vinyals, I. Babuschkin, W. M. Czarnecki *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, p. 350–354, 2019.
- [27] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 3438–3445, Apr. 2020.
- [28] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in *International Conference on Learning Representations*, 2021. [Online]. Available: <https://openreview.net/forum?id=i800PhOCVH2>
- [29] S. Zhang, Y. Liu, Y. Sun, and N. Shah, "Graph-less neural networks: Teaching old MLPs new tricks via distillation," in *International Conference on Learning Representations*, 2022.
- [30] P. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018. [Online]. Available: <https://arxiv.org/pdf/1806.01261.pdf>
- [31] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, 2008, pp. 466–474.
- [32] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: <http://jmlr.org/papers/v22/20-1364.html>
- [33] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "DEFO website," <https://sites.uclouvain.be/defo/>, 2015.
- [34] E. K. Çetinkaya, M. J. Alenazi, Y. Cheng, A. M. Peck, and J. P. Sterbenz, "On the fitness of geographic graph generators for modelling physical level topologies," in *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2013, pp. 38–45.
- [35] O. Karandin, F. Musumeci, O. Ayoub, A. Ferrari, Y. Pointurier, and M. Tornatore, "Quantifying resource savings from low-margin design in optical networks with probabilistic constellation shaping," in *2021 European Conference on Optical Communication (ECOC)*, 2021, pp. 1–4.
- [36] R. Hartert, P. Schaus, S. Vissicchio, and O. Bonaventure, "Solving segment routing problems with hybrid constraint programming techniques," in *Principles and Practice of Constraint Programming*, G. Pesant, Ed. Cham: Springer International Publishing, 2015, pp. 592–608.

4 | Data-Centric and Reliable Machine Learning for Fault Management

In this Chapter, we address several novel research problems related to training and deploying data-driven network fault identification models. Specifically, in Section 4.1, we consider the problem of constructing a “good-enough” dataset for training a ML-based fault classifier under two realistic assumptions: first, annotating new data is expensive; second, some failure classes are poorly represented in historical observations. We propose solving this problem through offline data augmentation and online Active Learning, resulting in practically significant performance gains and savings in annotation costs. Following up, in Section 4.2, we consider the problem of reliable fault classification, i.e., how to train a ML model with performance guarantees. Leveraging recent advances in the field of Conformal Prediction, we train classifiers that, given a telemetry data stream, output predictions as soon as their probability of misclassification is below a user-provided safety threshold. Numerical results on a real-world dataset demonstrate that our solution can provide both prompt and accurate predictions.

4.1. Data-Centric Machine Learning for Hardware Fault Classification

This paper considers the problem of classifying hardware faults of microwave network equipment based on telemetry data. This problem has been extensively investigated in prior literature, e.g., [54, 118, 119], with the state-of-the-art models reaching a classification accuracy comparable to human experts. In this paper, we adopt a different perspective on this problem. Specifically, we make the following claim: *fitting a model is easy, having to deal with data is hard*. While this insight might seem trivial, most of the literature takes the dataset for granted and instead focuses on designing ML models providing the highest possible performance. Instead, through multiple interactions with

our industrial partner, we realized that the main bottleneck to the project is curating a dataset of sufficiently high quality for training a model. To solve this problem, we proposed two solutions. First, before training a model, we proposed using generative models for augmenting a given training dataset with synthetic data. In the context of microwave networks, some failures might occur less frequently than others, and therefore, the ML model might be biased towards the best-represented classes. We numerically demonstrate that, in a scenario of extreme data imbalance, our proposed rebalancing strategy improves the performance of a state-of-the-art model. Second, after having trained a model, we proposed using Active Learning for leveraging the model's prediction to guide the annotation of new data. This is because annotating hardware failure data in microwave networks is expensive and time-consuming, since domain experts must manually inspect lengthy telemetry data. Our proposed approach demonstrates that, through Active Learning, we can train a top-performing model with a fraction of the data required by conventional random annotation. Our solutions were deployed in production, and gains aligned with the ones reported in the paper were observed by our industrial partner. Overall, we demonstrated that significant cost and performance gains can be achieved by reasoning about the quality of the dataset instead of over-engineering a model. Finally, we open sourced the dataset we utilized in our study, which was curated by domain experts in the microwave industry. To the best of our knowledge, this is the first public dataset of its kind, and we hope it will become a standard benchmark for future research in this field.

Machine Learning for Failure Management in Microwave Networks: a Data-Centric Approach

Nicola Di Cicco*, *Graduate Student Member, IEEE*, Memedhe Ibrahimimi*, *Member, IEEE*,
 Francesco Musumeci, *Senior Member, IEEE*, Federica Bruschetta, Michele Milano,
 Claudio Passera, and Massimo Tornatore, *Fellow, IEEE*

Abstract—We consider the problem of classifying hardware failures in microwave networks given a collection of alarms using Machine Learning (ML). While ML models have been shown to work extremely well on similar tasks, an ML model is, at most, as good as its training data. In microwave networks, building a good-quality dataset is significantly harder than training a good classifier: annotating data is a costly and time-consuming procedure. We, therefore, shift the perspective from a Model-Centric approach, i.e., how to train the best ML model from a given dataset, to a Data-Centric approach, i.e., how to make the best use of the data at our disposal. To this end, we explore two orthogonal Data-Centric approaches for hardware failure identification in microwave networks. At training time, we leverage synthetic data generation with Conditional Variational Autoencoders to cope with extreme data imbalance and ensure fair performance in all failure classes. At inference time, we leverage Batch Uncertainty-based Active Learning to guide the data annotation procedure of multiple concurrent domain-expert labelers and achieve the best possible classification performance with the smallest possible training dataset. Illustrative experimental results on a real-world dataset show that our Data-Centric approaches allow for training top-performing models with $\sim 4.5x$ less annotated data, while improving the classifier’s F1-Score by $\sim 2.5\%$ in a condition of extreme data scarcity. Finally, for the first time to the best of our knowledge, we make our dataset (curated by microwave industry experts) publicly available, aiming to foster research in data-driven failure management.

Index Terms—Microwave Networks, Machine Learning, Failure Management

I. INTRODUCTION

Machine Learning (ML) has found a broad set of applications in the design and management of modern communication networks [1]–[3]. Traditionally, failure management in microwave networks is dealt with via human inspection of telemetry data, which is an extremely time-consuming process. This requires hiring a team of trained domain experts who spend a significant amount of time identifying the causes of failure. In the specific context of microwave networks, modern ML-based approaches for failure management outperform static expert-driven decision rules, achieving close-to-human performance within a scalable decision-making framework [3].

*Nicola Di Cicco and Memedhe Ibrahimimi are co-first authors.

Nicola Di Cicco, Memedhe Ibrahimimi, Francesco Musumeci, and Massimo Tornatore are with the Department of Electronics, Information and Bioengineering (DEIB), Politecnico Di Milano, Italy. E-mail: {name}.{surname}@polimi.it

Federica Bruschetta, Michele Milano, and Claudio Passera are with SIAE Microelettronica, Cologno Monzese, Milan, Italy. E-mail: {name}.{surname}@siaemic.com

However, when training and deploying ML models for failure management, all network operators must face the challenge of building a good training dataset. For operators, the process of gathering and labeling data from the field is extremely expensive, as it requires continuous interaction with costly human annotators, causing significant time delays (e.g., for our dataset, it takes two full workdays for an annotator to label a hundred samples.)

Failure management in microwave networks is no exception to the aforementioned rule, as it requires *domain experts* to manually analyze and correlate the behavior of various link measures, i.e., Tx/Rx power and the corresponding failure alarms. Moreover, such experts must identify failure causes and failure types to take the *corresponding countermeasures* for restoring the disrupted services. Such countermeasures can imply something simpler, such as a decision to re-route traffic on another link until the quality-of-transmission in the original link is retrieved, or more complex decisions, such as sending a team in the field to identify and fix faulty equipment at a microwave site. While state-of-the-art ML-based approaches are known to automate such tasks with close-to-human accuracy, e.g., over 90% [3], the premise to reaching such performance is based on the assumption that there are sufficient labeled data for the ML model to learn robust failure identification rules.

In this work, in contrast to previous literature on failure management in microwave networks, we shift the perspective from *model-centric* (i.e., “How do we design a good ML model for failure identification?”) to *data-centric* (i.e., “How do we build a dataset of good quality for training ML failure classifiers?”). We opt for a *data-centric* approach motivated by the fact that, in practice, improving the quality of a dataset brings greater benefits to an ML model than engineering its architecture and hyperparameters [4]. We, therefore, explore two data-centric approaches for hardware failure identification in microwave networks, i.e., identify a failure cause that occurs in the hardware component of a microwave network. Specifically, we investigate the following data-centric Research Questions (RQs), addressing two crucial domain-specific challenges of ML-based failure identification in microwave networks: a) high annotation costs, and b) data scarcity due to infrequent failures.

RQ1) (Online phase) What is the optimal strategy for collecting and annotating data? To answer this question, we propose an Uncertainty-based Active Learning (UAL) approach. The objective of UAL is to leverage the predictive

uncertainty of the model for labeling the most informative samples only. By doing so, we can identify a small subset of data to be labeled by domain experts, thus leading to significant cost savings without compromising the state-of-the-art classification performance of our ML model.

RQ2) (Offline phase) How can we overcome data scarcity from infrequent failure classes? To deal with this challenge, we leverage data augmentation with synthetic data. Specifically, we propose using a Conditional Variational Autoencoders (CVAE) for generating synthetic samples with per-class, per-sample granularity. This is especially useful when the available dataset is highly imbalanced, i.e., in case a specific hardware failure presents itself only a few times during the data collection campaign. We illustrate that in a scenario characterized by extreme class imbalance, data augmentation with our CVAE ensures a fair classification performance for all hardware failure classes.

Last but not least, we make our dataset publicly available. To the best of our knowledge, ours is the first publicly available dataset comprising telemetry data from *real, in-production* microwave networks, collected over several years and painstakingly labeled by domain experts. We hope that our contribution will serve as a benchmark for future, exciting research on the microwave domain.

We summarize our key contributions as follows:

- We make a case for Data-Centric ML for failure management in microwave networks, and we focus on hardware failures (vs. mostly propagation failures in past work).
- We leverage the predictive uncertainty for quantifying the information gained by labeling batches of data points via Batch Uncertainty-based Active Learning (BUAL). Our approach allows training a top-performing model requiring only $\sim 15\%$ of the full dataset, thus granting proportional savings in annotation times and costs.
- We leverage CVAEs for generating synthetic data to enrich poorly represented failure classes, with the goal of improving the average classification performance among all failure classes.
- We make our high-quality dataset publicly available¹. Code is available upon request to the authors.

The remainder of the manuscript is organized as follows. In Section II, we discuss relevant related work. In Section III, we provide some background on microwave networks and describe the main characteristics of the dataset. In Section IV, we describe our proposed data-centric solutions and discuss how they fit in the context of failure management in microwave networks. In Section V, we discuss illustrative numerical results on our dataset. Section VI concludes the manuscript.

II. RELATED WORK

We overview recent literature investigating applications of ML to wireless networks, with emphasis on microwave networks. Additionally, we describe several recent works applying Active Learning (AL) and synthetic data generation in wireless networks.

¹<https://github.com/bonsai-lab-polimi/tnsm2024-data-centric>

A. ML for failure management in wireless networks

Failure management is a well-known problem in wireless networks, and it has been extensively investigated. Among the earliest works that address the problem of root-cause failure analysis in radio networks are [5] and [6]. Kliger *et al.* [5] have applied correlation techniques based on causality graphs and associate failure root causes to alarm sequence. Wietgreffe *et al.* [6] have adopted a neural network to correlate the presence of different alarms in mobile network equipment to an initial cause generating the alarm sequence. In [7], Szilágyi *et al.* designed a framework for automatic anomaly detection and cause identification in mobile networks, based on observations of alarms and employing a decision process that emulates human reasoning. Casas *et al.* [8] use decision trees for anomaly detection considering synthetically generated data drawn from realistic microwave network statistics. Ahmed *et al.* [9] used supervised learning to detect and localize failures in cellular networks, focusing on the observation of end-to-end service performance indicators. Wu *et al.* [10] proposed an anomaly detection framework considering time series for various performance indexes and applying a regression algorithm in cellular networks.

In recent years, our research group has actively investigated the application of ML-based approaches to microwave networks. Musumeci *et al.* [3] applied several ML approaches for automatic failure identification in microwave networks, in particular, proposing new models for the identification of failure causes by supervised ML and a semi-supervised automated labeling procedure based on auto-encoders. We address the problem of hardware failure as a supervised-learning classification problem as addressing similar problems in [3] via self-supervised pre-training was found to work overall poorly, therefore making the ground-truth information fundamental. Moreover, in microwave networks, different features can help capture different failure causes. For example, propagation failures can be discriminated from features associated with radio performance (e.g., transmitted/received power statistics and modulation format [3]), while equipment hardware failures can be discriminated from features associated with equipment alarms. Lateano *et al.* [11] and Ayoub *et al.* [12], [13] investigated several ML-based approaches for failure root-cause prediction and application of eXplainable Artificial Intelligence (XAI) in microwave networks, respectively.

Compared to previous model-centric works, we propose data-centric solutions allowing for cost-efficient data collection without compromising on the performance achieved in the state-of-the-art literature. Specifically, we leverage CVAEs for synthetic data generation and Batch AL to identify the most informative data points to label.

B. Active Learning applied to wireless networks

From an operator's point of view, gathering unlabeled data from the field is often a by-product of network monitoring. However, labeling said data (especially for hardware failures in microwave networks) can be an extremely costly and time-consuming process. To address the challenge of data labeling

with the objective of reducing operational cost and time, we leverage a Batch AL approach.

Pan *et al.* [14] make use of AL in microwave networks to continuously update the detection model at low cost. They show that, using AL, only 7% of the training data is required to achieve comparable results with the full dataset. Shahraki *et al.* [15] investigate the application of AL to query labels during training in the context of network traffic classification. Xu *et al.* [16] propose an AL-based solution to minimize the prediction error for classification-based mobile crowd-sensing subject to upload and query cost constraints.

In contrast to prior literature, we make a case for an AL framework leveraging the predictive uncertainty in tree ensembles. The motivation is twofold: first, tree ensembles are known to be the best-performing models in tabular datasets [17], [18]; second, tree ensembles allow for efficient epistemic (or knowledge) uncertainty sampling in AL, which is the current state-of-the-art for uncertainty sampling [19], [20]. Moreover, current AL applications in communication networks focus on querying one sample at a time, exacerbating annotation and training times. To solve this problem, we develop a Batch AL algorithm to simultaneously query up to 50 data points per iteration, showing minimal F1-Score degradation (~ 0.025 in the worst case) compared to the single-query optimum on the same annotation budget. This grants an order-of-magnitude reduction in annotation and model (re)-training times.

C. Synthetic data generation in wireless networks

A common challenge when applying ML to wireless networks is data scarcity, as it is often expensive to have access to operating networks, and, even when access is granted, it is expensive to gather and label data. Most existing research relies on mathematical models for data generation, however, such models are based on several assumptions and simplifications and may not fully represent realistic scenarios.

In the following, we cover related works that have applied Generative Adversarial Networks (GANs), Synthetic Minority Oversampling Technique (SMOTE), and Variational Auto-Encoders (VAEs) to address data scarcity.

Ayanoglu *et al.* [21] provide an overview of applying GANs to next-generation networks, and show several case studies of synthetic data generation. Navidan *et al.* [22] review the application of GANs to computer and communication networks, including mobile networks, network analysis, the internet of things, the physical layer, and cyber-security. Yang *et al.* [23] make use of GANs for autonomous wireless channel modeling without any domain-specific knowledge or technical expertise.

Clark IV *et al.* [24] investigate the problem of data augmentation in the context of radio frequency systems. Davaslioglu *et al.* [25] investigate the application of GANs to generate additional synthetic data to improve classifier accuracy and adapt training data to spectrum dynamics applied to spectrum sensing. Tang *et al.* [26] investigate the application of GANs to automated modulation classification in cognitive radio networks and evaluate the classifier's performance when the dataset is enriched with synthetically generated data.

An alternative, simpler approach for synthetic data generation is SMOTE. Massaoudi *et al.* [27] utilize it to handle class imbalance for intrusion detection in smart grid operations. Sun *et al.* [28] utilize SMOTE to address class imbalance in the context of self-organizing cellular networks to address the problem of failure diagnosis. Abdulkareem *et al.* [29] address class imbalance in an Internet-of-Things (IoT) context to perform a classification task for intrusion detection. Similarly, Tesfahun *et al.* [30] has implemented SMOTE to address class imbalance to perform an intrusion detection classification task using a Random Forest classifier.

Several works have used VAEs for generating synthetic data. Razghandi *et al.* [31] propose a VAE-GAN as a smart grid data generative model able to learn several types of data distributions and generate samples from the same distributions without performing any prior analysis on the data before the training phase. Qu *et al.* [32] propose a network data reinforcement method based on the multiclass VAEs to complete training tasks with a limited amount of data. The proposed solution can control the proportion of different classes by adjusting parameters, thus solving the imbalance problem in network data. Khan *et al.* [33] use VAEs for data augmentation for failure management in optical networks.

Compared to previous works that make use of other approaches for synthetic data generation, such as Conditional Tabular Generative Adversarial Networks (CTGANs) [34] and SMOTE [35], a CVAE-based solution proves to be more appropriate for our purpose. GAN-based solutions are unsuitable for our use case because i) GANs thrive in the presence of large volumes of data, but in our case, the initial dataset is of modest size (i.e., less than 1700 observations), and ii) GANs require a delicate balance between the generator and discriminator networks, and in small datasets this balance can be harder to achieve, leading to training instability, oscillations, and difficulties in convergence. We opted for VAEs as we empirically found them more reliable and easier to train than GANs for our application. We underline that the empirical studies of [36] illustrate different pros and cons of using VAEs as opposed to GANs for tabular data generation. VAEs achieve better performance metrics and are easier to train, but GANs make it easier to achieve differential privacy for the training data (which is presently not a concern in this work.) As for simpler approaches such as SMOTE, there are several shortcomings [37], such as: i) in case a minority class is extremely sparse, SMOTE could lead to overfitting and to a synthetic data generation that does not adequately represent the underlying distribution of the real data; ii) in high-dimensional spaces, SMOTE's effectiveness decreases as the curse of dimensionality makes it difficult to find meaningful neighbors for interpolation.

We do not claim that CVAE-based approaches are always better than other alternatives, e.g., GANs, as such choice depends on the case study. However, for our use case study, we show that adding synthetically generated data improves the per-class performance, especially for the under-represented failure classes. Compared to the related works for data generation, namely, CTGAN and SMOTE, we provide a qualitative comparison in the numerical results (see Section V).

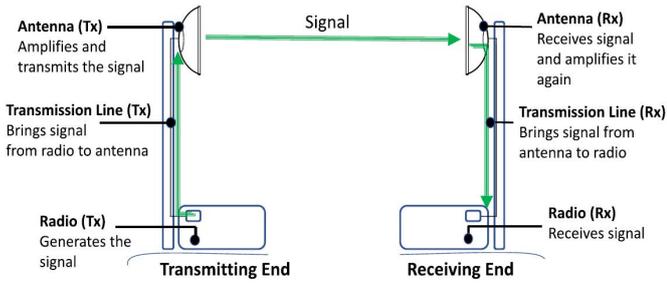


Fig. 1. Basic components of a microwave link [3]

III. BACKGROUND ON MICROWAVE NETWORKS AND DATASET DESCRIPTION

In this Section, we provide a brief overview of the main components in a microwave network, and we describe the SIAE Microelettronica hardware failures dataset.

A. Microwave Networks

Figure 1 shows the basic structure of a microwave link, highlighting the transmitting end and the receiving end. Each end is composed of three main elements:

- 1) A Microwave radio that generates the signal (at the TX site) and receives the signal (at the RX site). The microwave radio can be placed at different locations, i.e., either inside a building or shelter (full-indoor), in the proximity of the antenna (full-outdoor), or by adopting a hybrid solution, called split-mount, where the electronic devices are distributed between an Outdoor Unit (ODU) and an Indoor Unit (IDU).
- 2) A transmission line brings the signal from/to radio to/from the antenna. It is typically implemented via coaxial cables, suitable for frequencies up to around 2 GHz, or waveguides, used for higher frequencies up to around 13 GHz (and, in some rare cases, up to 40 GHz). Transmission lines are responsible for non-negligible signal losses, depending on signal frequency, and may strongly affect the quality of transmission in case of physical medium deterioration.
- 3) A directional antenna, usually parabolic-shaped, is characterized by its gain, size, and directivity functions.

In this paper, we focus on hardware failures that can impact the hardware radio, i.e., IDU failure, ODU failure, cable failure, and power failure. In the following, we provide a detailed description of our hardware failures dataset.

B. SIAE Microelettronica hardware failures dataset

Microwave-link unavailability is typically defined in terms of *Unavailability Seconds* (UAS), representing the number of seconds over which the number of errored bits exceeds a given threshold. The UAS can be caused by various phenomena, such as *propagation failures*, due to, e.g., atmospheric factors such as rain, fog, temporary obstacles, or *hardware failures* due to equipment malfunction or aging. Hence, to automate the identification of failure causes leading to UAS and to avoid the continuous involvement of domain experts, we develop a

set of ML-based solutions. To this end, we constructed a high-quality dataset of hardware failures from a real microwave network via a Network Management System developed by SIAE Microelettronica.

Our dataset comprises four non-overlapping classes of failure types: 1) *Class-0*: IDU failure, 2) *Class-1*: ODU failure, 3) *Class-2*: cable failure, and 4) *Class-3*: power failure. Each failure type is identified based on alarms issued by the radio equipment, serving as input features to the ML-based classifier.

We collected and labeled 1669 data points (observations) in total. The ground-truth labels were attributed and extensively validated by domain experts. Specifically, the frequency of each failure class in our dataset is as follows:

- Class-0 (IDU failure): 515 observations
- Class-1 (ODU failure): 611 observations
- Class-2 (Cable failure): 207 observations
- Class-3 (Power failure): 336 observations

The skew of our dataset, i.e., the ratio between the highest and the lowest class frequencies, is approximately 3, showing a moderate degree of class imbalance.

Each observation in the dataset aggregates data from 15-minute non-overlapping windows. For each microwave link and 15-minute window, the input to the classification problem consists of 164 features, each one corresponding to the number of seconds in which a specific alarm was active in the window. Specifically, the features take values between 0 and 900, with 0 indicating that the alarm is *OFF* and 900 indicating that the alarm is *ON* during the whole 15-minute window. To provide a rough idea of the human effort required for the manual data labeling, two domain experts from SIAE Microelettronica have spent more than 2 weeks² to label all 1669 data points.

To preserve confidentiality, we have performed the following operations to anonymize the dataset: 1) We masked all alarm names to [alarm₀ - alarm₁₆₃], 2) We removed all timestamps, 3) We randomly permuted the rows and columns of the original dataset (i.e., there is no semantic correlation between adjacent columns and no temporal correlation between adjacent rows), 4) We removed all sensitive information that could be used for tracing the specific microwave link.

For both the synthetic data generation and AL, we initially perform a *feature preprocessing* step, and then we estimate the performance of several ML models in terms of F1-score via stratified K-fold cross-validation.

C. Feature preprocessing

We devise three strategies for representing the alarm features based on when an alarm in a link is ON:

- 1) *Mode-1: Binary*, in which alarms are binarized, i.e., “0” if an alarm is OFF for the full 15-minute window, and “1” if the alarm is ON for at least 1 second.
- 2) *Mode-2: Categorical*, in which alarms are represented by categorical features.³ We consider four categories: “0” if an alarm is OFF for the full 15-minute window, “1”

²For our specific task, domain experts were able to annotate approximately 50 data points for each full workday.

³Note that the identification of categories is done in collaboration with domain experts, to distill the most useful information for decision-making.

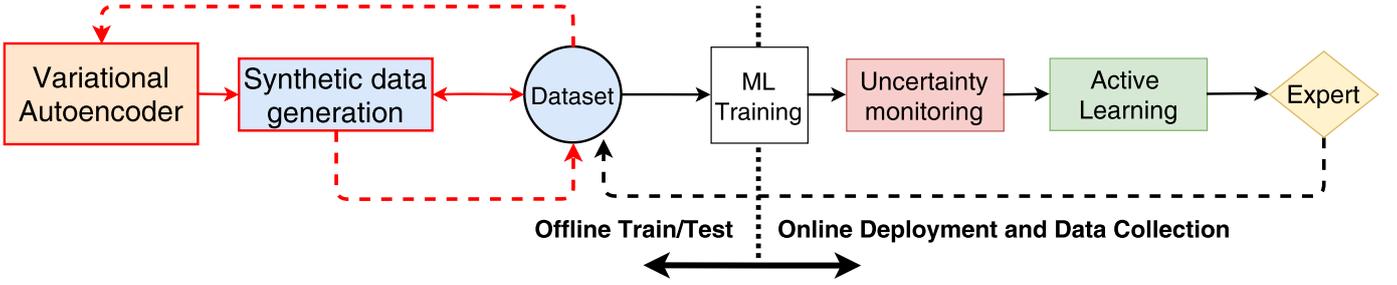


Fig. 2. Data-Centric ML for failure management in microwave networks. Synthetic data generation is performed in the *Offline Train/Test* phase, while AL is leveraged for the *Online Deployment and Data Collection* phase.

if an alarm is ON for no more than 45 seconds (low-severity alarm or false alarm), “2” if an alarm is ON between 45 seconds and 450 seconds (medium-severity alarm), and “3” if an alarm is ON between 450 seconds and 900 seconds (high-severity alarm).

- 3) *Mode-3: Inherent*, in which alarms take values between 0 and 900, as measured by the radio equipment.

In our numerical results, while we consider all three modes of data representation in the case of UAL, we consider only Mode-1 and Mode-2 features for synthetic data generation via the CVAE, as we have observed that Mode-3 (i.e., not applying any preprocessing) results in worse classification performance. Moreover, we have found the synthetic data generation process to be highly unreliable when generating data with continuous features, compared to either categorical or binary features. Overall, as a generic guideline for all the ML tasks considered in this work, binning the continuous alarm features turns out to be consistently beneficial.

IV. DATA-CENTRIC ML FOR FAILURE MANAGEMENT IN MICROWAVE NETWORKS: BUILDING BLOCKS

In Fig. 2 we illustrate the core elements in our Data-Centric ML solution for failure identification in microwave networks. Our solution comprises two sections: 1) Offline Train/Test, and 2) Online Deployment and Data Collection. Synthetic data generation is performed during the training phase when an offline dataset is already available. AL is performed during real-time inference, in order to guide the data labeling process towards the most informative samples.

In the following, we provide a detailed description of each building block. We first discuss UAL, as the microwave network we consider is currently operating, and then comment on synthetic data generation, as this process can be performed only once a dataset is available for offline training.

A. Uncertainty-based Active Learning

The goal of AL is to train an ML model with the fewest amount of labeled samples. Our core assumption is that large quantities of unlabeled data are available. This holds true for failure management in microwave networks, as collecting alarm sequences is fully automated and inexpensive. As mentioned before, annotating data in our application scenario is not only very costly but also time-expensive.

At a high level, AL allows the model to actively *query* the ground-truth label from unlabeled samples. The most crucial

design choice in AL is defining a query strategy for selecting only the most informative samples.

A simple way to quantify the informativeness of an unlabeled alarm set is to estimate the predictive uncertainty associated with its predicted hardware failure class. Generally, the total predictive uncertainty can be decomposed into *Data Uncertainty* (DU) and *Knowledge Uncertainty* (KU) [38]. DU is caused by the inherent noise in the data, and therefore cannot be reduced by increasing the dataset size. On the other hand, KU is caused by the model’s ignorance, i.e., by the model being presented with data from a different distribution than training. In contrast to DU, KU may be reduced by increasing the dataset size. Therefore, we are interested in leveraging KU as an “informativeness score” for labeling new samples.

While there are several strategies for uncertainty decomposition in ML, e.g., Bayesian Neural Networks, our previous works [3], [12] illustrated that the best-performing ML models for failure identification in microwave networks were ensembles of Decision Trees (DTs), such as Random Forests (RFs) or Gradient Boosting Decision Trees (GBDTs). Indeed, the superiority of tree-based models on small to medium-size tabular datasets has been validated by large-scale empirical studies [17], [18]. Therefore, we would like to seamlessly incorporate AL in our failure management framework while keeping our best models. For this reason, we leverage approximate uncertainty decomposition in ensemble models, such as RFs and GBDTs.

Formally, assume that we have trained an ensemble of M models on a dataset \mathcal{D} . Let $p(y|\mathbf{x}, \mathcal{D})$ be the predictive distribution of the ensemble, $p(y|\mathbf{x}, h^{(m)})$ be the predictive distribution of m -th model h in the ensemble (e.g., in RFs and GBDTs $h^{(m)}$ is a DT), and let $p(h|\mathcal{D})$ be the distribution of the ensemble members if trained on dataset \mathcal{D} (e.g., in RFs and GBDTs this represents the probability distribution of generating a certain DT). \mathbf{x} denotes the set of alarms while y denotes the failure class. We can define and estimate the KU in Eq. (1), as follows:

$$\underbrace{\mathcal{I}(y, h|\mathbf{x}, \mathcal{D})}_{\text{Knowledge Unc.}} = \underbrace{\mathcal{H}[p(y|\mathbf{x}, \mathcal{D})]}_{\text{Total Unc.}} - \underbrace{\mathbb{E}_{p(h|\mathcal{D})} \mathcal{H}[p(y|\mathbf{x}, h)]}_{\text{Data Unc.}} \quad (1)$$

$$\approx \mathcal{H}\left[\frac{1}{M} \sum_{m=1}^M p(y|\mathbf{x}, h^{(m)})\right] - \frac{1}{M} \sum_{m=1}^M \mathcal{H}[p(y|\mathbf{x}, h^{(m)})],$$

where $\mathcal{H}[\cdot]$ and $\mathcal{I}[\cdot, \cdot]$ denote Shannon entropy and Information Gain, respectively [38]. The *Total Uncertainty* (TU) is defined

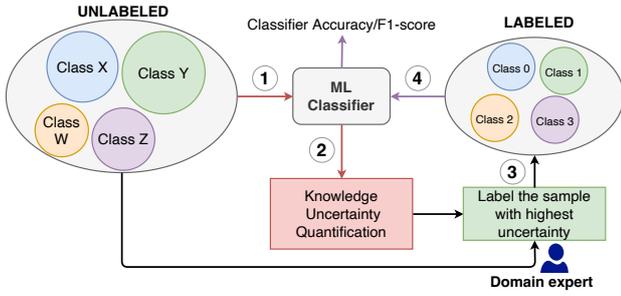


Fig. 3. Single-query Uncertainty-based Active Learning (UAL)

as the entropy of the predictive distribution of the ensemble, obtained by averaging between all ensemble members.⁴ The DU is defined as the expected entropy value of the single ensemble member. Finally, the KU is defined as the information that model h gains by revealing the failure class y associated with the alarm set x , and is estimated by subtracting the DU from the TU.

We develop two approaches based on the policy for querying data points to label: i) single-query UAL, and ii) multi-query BUAL. Figure 3 and Fig. 4 illustrate the application of UAL and BUAL, respectively, for guiding the data collection and labeling procedure in our failure management framework. We assume a small initialization pool of labeled data, e.g., 20-40 labeled alarm sets, and a large pool of unlabeled data, e.g., 1000-1500 alarm sets. In the following, we describe the workflow of UAL and BUAL.

B. Single-query Uncertainty-based Active Learning

The framework operates in four steps (see Fig. 3): (1) Data from the pool of *UNLABELED* data are queried via the *ML classifier* which then does (2) *Uncertainty Quantification* of data points in terms of KU. The alarm set with the highest KU values is forwarded to a domain expert to be labeled (3) *Label the sample with highest KU*, which is then added to the pool of labeled alarms. Note that in the case of single-query UAL, one sample is labeled and added to the pool of *LABELED* data. The ML model is retrained from scratch on the updated dataset (4), and the process is iterated until a stopping condition is reached (e.g., a pre-defined budget of data points to label is met or a pre-defined performance metric, such as F1-Score, is met on a held-out validation dataset). In our case, we consider a fixed budget of data points, which can be related to a limited financial budget for compensating domain experts and/or to a time constraint on data collection and labeling.

C. Batch Uncertainty-based Active Learning

In principle, UAL assumes adding one data point per iteration, which guarantees that, at each iteration, we are sampling the alarm set that provides the highest expected information gain for our model. We now consider a Batch AL scenario, where multiple data points are labeled per AL iteration. There are two main practical reasons motivating the

⁴In ensemble learning, the KU can also be interpreted as the level of disagreement between different ensemble members.

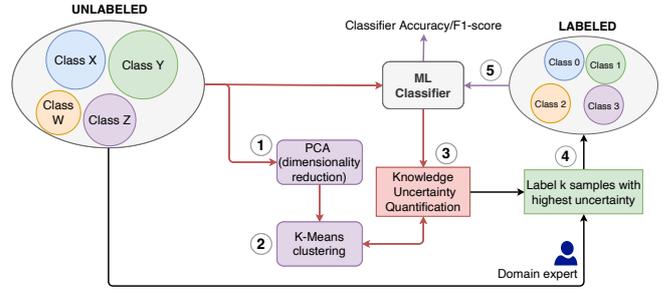


Fig. 4. Batch Uncertainty-based Active Learning (BUAL)

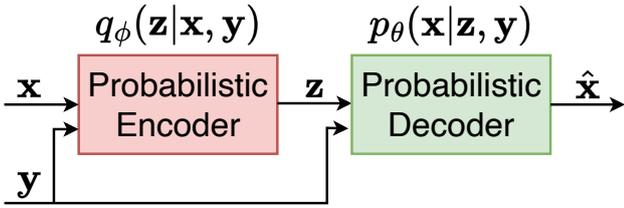
need for Batch AL over single-query AL: 1) in real-world scenarios, it is often the case that multiple domain experts are dedicated to data annotation; 2) when annotating one data point per iteration, it is necessary to re-train the ML classifier for each newly-labeled data point. Instead, Batch AL re-trains the ML classifier only once per batch of X data points, e.g., $X = 50$, granting a proportional reduction in training times. We therefore propose BUAL to solve the aforementioned issues.

A first, naive query strategy for BUAL is to take the top K data points with the highest KU. We refer to this simple approach as *BUAL-TopK*. However, the top K data points with the highest KU might be highly correlated: for instance, revealing the ground truth for one point only would render all the remaining $K - 1$ not very informative. Indeed, our numerical results in Section V-B illustrate that *BUAL-TopK*, though performing much better than random sampling, is considerably less data-efficient than single-query UAL.

Therefore, we aim for a Batch AL algorithm able to sample points that are both i) highly informative and ii) sufficiently diverse. To balance this delicate trade-off, we leverage dimensionality reduction and K-means clustering to develop *BUAL-KMeans*, which, unlike *BUAL-TopK*, allows multiple simultaneous queries (up to 50) with negligible performance degradation with respect to single-query UAL.

The core design choice in *BUAL-KMeans* is to leverage K-means clustering [39] to avoid sampling redundant data points. Previous literature in Batch AL with neural networks illustrated that clustering is an effective strategy for balancing diversification and informativeness [40], [41]. Unfortunately, neural networks are a bad fit for our problem, as it is formulated on tabular data [17], [18]. Moreover, given the relatively high dimensionality of our dataset (164 features) and its moderate size (1669 observations) with respect to the large-scale benchmark datasets in deep learning, distance-based clustering algorithms such as K-means cannot be used out-of-the-box for our problem.

To address these issues, we apply a dimensionality-reduction step to the unlabeled data using Principal Component Analysis (PCA) [42] and keeping only the first three dimensions with the highest variance. This allows filtering out small pairwise distances, enabling effective data separation via K-means clustering. Note that, even though the dimensionality reduction causes significant information loss, the compressed features are used only for clustering, whereas the ML classifier is presented with the full set of alarm features.



$$\mathcal{L}_{\text{CVAE}} = -\log p_{\theta}(\mathbf{x}|\mathbf{z}, \mathbf{y}) + D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}, \mathbf{y}) \parallel p(\mathbf{z}|\mathbf{y}))$$

Fig. 5. Conditional Variational Autoencoder (CVAE). The one-hot encoded hardware failure class vector \mathbf{y} conditions both the encoder and the decoder.

Putting everything together, the BUAL-KMeans algorithm operates in five steps (see Fig. 4): (1) Data from the pool of *UNLABELED* data are fed to the *PCA* block for dimensionality reduction, and then, in step (2), the unlabeled data are clustered via *K-Means Clustering*. In step (3), the *KU Quantification* estimates the information gain through the *ML classifier* for all data points in the pool of *UNLABELED* data. The decision on which data points to pass on to the domain experts is taken based on K-Means Clustering, such that the data point with the highest KU from each cluster is passed on, e.g., we set $K = 10$ clusters for selecting 10 data points to be labeled in one iteration. In step (4), the batch of alarm set with the highest KU values is forwarded to domain experts for labeling (*Label k samples with highest uncertainty*) and then added to the pool of *LABELED* data. Similarly to single-query UAL, in step (5), the ML model is retrained from scratch on the updated dataset, and the process is iterated until a stopping condition is reached.

D. Synthetic Data Generation with CVAEs

VAEs are deep generative neural networks that can efficiently learn probabilistic models of high-dimensional data [43]. A VAE comprises a probabilistic encoder network and a probabilistic decoder network. The encoder $q_{\phi}(\mathbf{z}|\mathbf{x})$, parameterized by ϕ , maps an input data sample to a latent space. The decoder $p_{\theta}(\mathbf{x}|\mathbf{z})$, parameterized by θ , maps a sample from the latent space to the input space. VAEs are trained by minimizing the Evidence Lower Bound (ELBO) loss, given by Eq. (2), as follows:

$$\text{ELBO} = \underbrace{-\log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{MLE reconstruction}} + \underbrace{D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))}_{\text{KL regularization}}. \quad (2)$$

The ELBO loss comprises a Maximum Likelihood Estimation (MLE) reconstruction term and a Kullback-Leibler (KL) regularization term [43]. The MLE reconstruction term ensures that the samples reconstructed by the decoder are faithful to the original distribution of the input data. The regularization term forces the variational posterior to be as close as possible to the true posterior. In practice, the KL regularization term forces the latent space to assume the shape of the prior distribution. In this way, after the VAE is trained, one can efficiently generate synthetic data by i) sampling \mathbf{z} vectors from the prior distribution and ii) feeding the \mathbf{z} samples to the probabilistic decoder to produce synthetic \mathbf{x} samples.

While one could use vanilla VAE to generate data, we are interested in sampling from the class-conditioned distribution of the input data. In our application scenario, we would like to have fine-grained control over the number of samples we generate for each failure class, as some classes may be severely underrepresented. While sampling from a specific class could be trivially achieved by training a vanilla VAE and performing rejection sampling, there are two major drawbacks. First, said methodology can become computationally expensive if large amounts of per-class data are queried and/or some classes are less represented in the input data, and are therefore generated less frequently. Secondly, it is well-known that VAEs suffer from mode collapse, that is, the model generates repetitive samples that do not capture the full diversity of the training data. Specifically, in our application scenario, we are interested in modeling 1) the diversity between alarms signaling *different* hardware failure classes, and 2) the diversity of alarms signaling *the same* hardware failure class.

To solve these problems, we employ CVAEs, which learn a probabilistic model of the input data conditioned on some contextual information (in our case, the class information). This is easily achieved by supplying a one-hot vector \mathbf{y} encoding the hardware failure class to both the encoder and the decoder networks. The CVAE ELBO loss [44] then becomes as given by Eq. (3), as follows:

$$\text{ELBO} = \underbrace{-\log p_{\theta}(\mathbf{x}|\mathbf{z}, \mathbf{y})}_{\text{MLE reconstruction}} + \underbrace{D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}, \mathbf{y}) \parallel p(\mathbf{z}))}_{\text{KL regularization}}. \quad (3)$$

Similarly as before, to generate new data i) we sample \mathbf{z} vectors from the prior distribution, ii) we feed the \mathbf{z} samples and a class vector \mathbf{y} to the probabilistic decoder. In this way, we achieve fine-grained control on the exact per-class quantity of synthetic data we generate with no additional computational effort. Moreover, as the class information is supplied to the model, the CVAE must only model the diversity between samples in the same class, which is arguably an easier learning task compared to a standard VAE.

Figure 5 shows the building blocks of the CVAE used for augmenting our hardware failures dataset, highlighting that *Probabilistic Encoder* and *Probabilistic Decoder* are conditioned on the one-hot encoded class \mathbf{y} .

V. ILLUSTRATIVE NUMERICAL RESULTS

In this Section, we first describe the evaluation settings and then discuss our illustrative numerical results for AL and synthetic data generation.

A. Evaluation settings

Here, we discuss our numerical evaluations of the proposed Data-Centric methodologies for hardware failure identification in microwave networks.

We report macro-averaged F1-score (hereafter referred to as macro F1-score) as the main performance metric, as we are interested in fair performance for all classes.⁵ As UAL requires

⁵In our simulations, we consider the macro-averaged F1-score, which is computed using the arithmetic mean of all per-class F1-scores. This metric treats all classes equally regardless of their number of data points in the dataset.

retraining the model after a single query, we also report the model training times. Our experiments have been conducted on a Linux PC with an Intel Core i7-6700 processor and 32 GB RAM. Our ML algorithms were implemented using Scikit-Learn [45] for RFs and XGBoost [46] for GBDTs. We opted to use XGBoost and RFs for our application because they have been found to be among the best-performing models for tabular data [17], [18].

B. Uncertainty-based Active Learning

We consider three modes of alarm feature representation and three sizes of the initial dataset.

- Alarm representations are described in Section III, and are: 1) Mode-1: Binary, 2) Mode-2: Categorical, and 3) Mode-3: Inherent.
- We consider three cases of initially labeled datasets: 1) 25 data points, 50 data points, and 3) 100 data points. The number of data points per class in the initially labeled dataset reflects the percentage of each hardware failure class in the complete dataset, i.e., Class-2 comprises 12% of the total dataset, so in 100 data points, 12 data points belong to Class-2.

We evaluate the classification performance in terms of macro F1-score using stratified K-fold cross-validation with $K = 5$. We assume an available budget for labeling 500 data points, which, in our practical experience, translates to a whole week of work for two domain experts.

Note that all numerical evaluations for data labeling, i.e., the proposed UAL and all baselines, are conducted considering the original dataset described in Section III-B, with no additional synthetically generated data involved.

For Batch AL, we consider querying *i*) 1 sample, *ii*) 10 samples, *iii*) 25 samples, and *iv*) 50 samples per AL iteration.

We consider three baselines: 1) Virtual Ensembles in Gradient Boosted Decision Tree (GBDT-VE) models as a representative state-of-the-art baseline [38] for uncertainty quantification and Active Learning on tabular datasets. GBDT-VE constructs a “virtual ensemble” by aggregating the predictions of multiple sub-models that compose a single GBDT model. The sub-models are constructed via truncation, i.e., by removing trees from the starting model. In contrast to RFs, where trees are trained independently, GBDT-VE has a high degree of correlation between trees, which adversely affects its uncertainty estimations. 2) Random sampling, i.e., data points to be queried are selected randomly, corresponding to a standard labeling strategy without AL, and 3) Full-Training, i.e., the performance of the best ML classifier when training on the full dataset as an upper bound on the best performance a ML model can achieve. For our problem, RFs and GBDTs roughly achieve the same level of performance when trained on the full dataset.

1) *UAL vs. Baselines*: Figures 6a, 6b and 6c show results for the RF classifier in terms of macro F1-score for Mode-1 with K-queries per AL iteration, where $K = 1, 10, 25$ and 50.

We observe that single-query UAL meets the Full training F1-score by querying less than 200 data points, compared to the approximately 1400 data points used for Full training. In

TABLE I
SINGLE-QUERY UAL TOTAL TRAINING TIMES AS A FUNCTION OF 1) CLASSIFIER TYPE, AND 2) ALARM MODE REPRESENTATION.

Classifier	Alarm mode	Training time (mins)
GBDT	BINARY	420
	CATEGORICAL	179
	INHERENT	234
RF	BINARY	19
	CATEGORICAL	19
	INHERENT	19

other words, UAL would require two domain experts to work for two days instead of two weeks. These remarkable time (and hence cost) savings have proven to be very significant for our industrial collaborators for a timely and cost-efficient deployment of the proposed ML-based failure management framework. Moreover, single-query UAL always outperforms GBDT-VE and Random Sampling, the latter being unable to meet the Full training F1-score even after labeling 500 additional data points to the training set. This highlights the practical effectiveness of AL for our application scenario.

In the Batch AL setting, we observe that BUAL-TopK, while outperforming random sampling, is significantly less data-efficient than single-query UAL and GBDT-VE. This is because BUAL-TopK does not consider diversifying the data in the batch. Conversely, BUAL-KMeans follows almost perfectly the trend of single-query UAL for batch sizes $K=10$ and $K=25$, while performing comparably to single-query GBDT-VE for $K=50$.

The practical advantages of BUAL over single-query UAL are exemplified in Fig. 6d, which shows the F1-score as a function of the AL iterations for a batch size $K=50$. We observe that BUAL-KMeans and BUAL-TopK reach the F1-score of Full training with at most 10 AL iterations, whereas single-query UAL requires around $\sim 25x$ more AL iterations to reach the same F1-score. In practical terms, the cost savings of BUAL-KMeans over single-query UAL are twofold: first, if multiple domain experts are available, the annotation times can be reduced proportionally to the batch size; second, BUAL-KMeans drastically reduces the number of times the model needs to be re-trained from scratch.

We have performed the same analysis for the other modes of alarm features, namely, Mode-2 (Categorical) and Mode-3 (Inherent), and the main takeaways are consistent with the results shown above. Similarly, the same takeaways highlighted in previous paragraphs hold in case the initially labeled data set has 50 data points and 100 data points.

2) *Training times*: In the following, we briefly discuss the training times of single-query UAL while varying: 1) the two ML classifiers considered in our work, i.e., RF vs GBDT, and 2) alarm feature modes, i.e., Binary vs Categorical vs Inherent.

We report the training time in the case of UAL single-query, i.e., querying one data point per UAL iteration, as we observed that Batch AL with batch size K follows roughly a proportional reduction in training time by $\frac{1}{K}$. Batch AL also includes PCA and clustering, even though their contribution to the total training time is negligible. Note that, for a scenario with a budget of 500 data points to be labeled, there are 500 iterations of the AL framework.

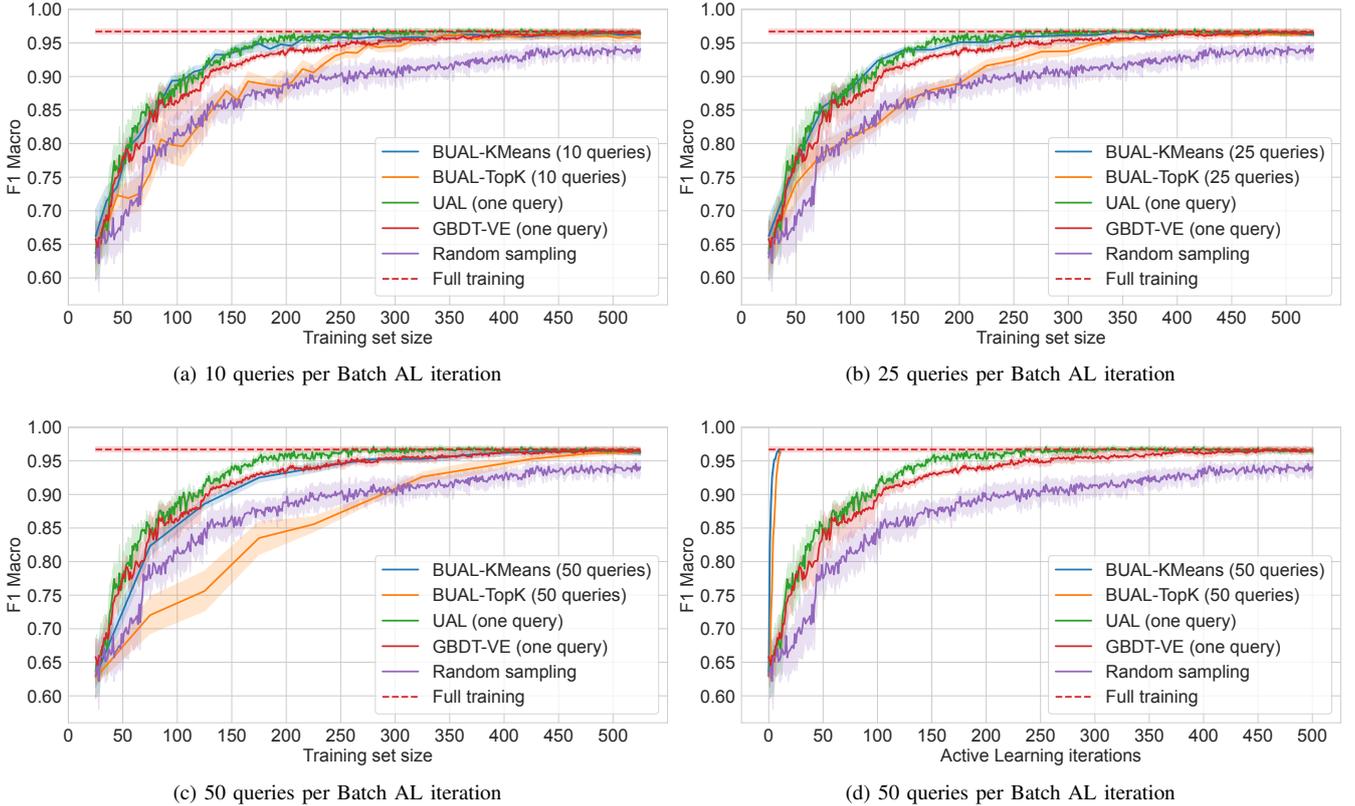


Fig. 6. F1-score of an RF classifier for our proposed AL strategies, UAL and BUAL, against GBDT-VE, Random sampling, and Full training. (a)-(c): F1-score while varying the number of concurrent AL queries. BUAL-KMeans performs nearly as well as single-query UAL. (d): F1-score as a function of the number of AL iterations for 50 queries per batch. BUAL requires very few AL iterations to reach the Full training upper bound.

Table I compares the training time for GBDT and RF. We observe that RFs are significantly more computationally efficient than GBDTs (19 vs. 420 minutes). This is because in RFs the KU can be computed considering each tree in the ensemble, while with GBDTs one needs to train multiple GBDT models, due to the fact that boosted trees are too highly correlated. Though GBDT-VE is more computationally efficient than GBDT ensembles in computing KU, we demonstrated that, for our use case, its AL performance is generally inferior compared to our RFs.

Regarding alarm modes, we notice that the training time of RFs is similar regardless of the alarm mode, while in the case of GBDT we observe that the training times are highly dependent on the chosen alarm representation. Specifically, the Categorical mode has the lowest training time, followed by the Inherent and Binary modes. This behavior is purely dependent on the implementation details of XGBoost, especially on the tree-growth policy. A possible explanation could be that, since Binary features are less informative than Categorical features, XGBoost tends to grow more trees to their maximum allowed depth (6 by default), resulting in longer training times.

Finally, we consider that a total training time of a few tens of minutes is reasonable in practice, as SIAE Microelettronica has adopted our solution in the field.

C. Synthetic data generation

We now illustrate our numerical results for synthetic data generation. To simulate a scenario of extreme data scarcity,

we randomly remove 80% of the data points from the training set for Class-2 (i.e., the least represented class), leading to an extreme imbalance in the class representation. This mimics a practical scenario of a rare occurrence of specific failures during the data collection campaign. The goal of this analysis is to show that, even in the case of a highly imbalanced dataset, data augmentation with CVAE allows for a fair classification performance for all hardware failure classes. We consider three dataset rebalancing strategies via CVAE:

- 1) **Balanced**: starting from an unbalanced scenario, generates synthetic data to ensure an equal number of data points for all classes.
- 2) **Balanced x2**: doubles the number of data points once having balanced the number of data points for all classes.
- 3) **Fixed**: generates a fixed number of synthetic data for each class, regardless of their percentage in the total dataset. We consider two cases of adding a fixed number of synthetic data: 1) add 150 data points to each class, and 2) add 400 data points to each class.

We compare the following models: 1) **Base model**: the ML model is trained on real data only, and 2) **Mix model**: the ML model is trained augmenting the real data with synthetic data. For the sake of brevity, we only represent results with Mode-1: Binary alarms. Similar considerations can be drawn for the Categorical representation of alarms.

Note that as *Base* considers training on real data only, the size of the training set is always equal to 1204 data points. On the other hand, the total number of training data in *Mix*

TABLE II
 SYNTHETIC DATA GENERATION: ACCURACY, F1-SCORE, PRECISION AND RECALL FOR **BASE** AND **MIX** MODELS COMPARING CVAE, SMOTE AND CTGAN. FOR CVAE, WE REPORT FOUR CASES OF DATA GENERATION: I) BALANCED, II) BALANCED X2, III) FIXED 150, AND IV) FIXED 400. WE REPORT MEANS AND STANDARD ERRORS OVER 5-FOLD CV.

Metric	Base	Balanced	Balanced x 2	Fixed 150	Fixed 400	SMOTE	CTGAN
Accuracy	91.49 (1.03)	93.05 (0.81)	92.39 (0.89)	92.45 (1.01)	92.75 (1.34)	90.71 (1.03)	91.73 (1.11)
F1-score	88.13 (1.31)	91.01 (1.02)	89.97 (0.96)	90.08 (1.31)	90.38 (1.72)	86.48 (1.35)	88.85 (1.46)
Precision	92.57 (1.13)	92.91 (0.87)	92.16 (1.05)	92.74 (1.27)	92.98 (1.54)	90.97 (1.14)	91.23 (1.17)
Recall	86.40 (1.32)	89.85 (1.12)	88.78 (0.94)	88.69 (1.29)	89.00 (1.78)	85.06 (1.31)	87.77 (1.48)

varies depending on the amount of synthetic data added. In particular, for *Balanced*, the total number of training data is 1956 data points, for *Balanced x 2*, the total number of training data is 3904 data points, for *Fixed 150*, the total number of training data is 1804 data points, and for *Fixed 400*, the total number of training data is 2804 data points.

We consider two state-of-the-art baselines, namely, CTGAN and SMOTE, that add synthetic data to the training set and balance the per-class representation. Therefore, in terms of the number of data points in the training set, they correspond to the *Balanced* scenario.

We compare all approaches (CVAE, SMOTE, and CTGAN) in case of *Base* and *Mix* in terms of i) Accuracy, ii) macro F1-score, iii) Precision, and iv) Recall (see Table II), and per-class F1-score (Table III) to quantify the benefit of augmenting real data with synthetic data. We report means and standard errors over 5-fold cross-validation.

In the following, we first compare the different flavors of CVAE, i.e., *Balanced*, *Balanced x2*, *Fixed 150*, and *Fixed 400*. We then compare the best-performing CVAE to two state-of-the-art baselines: CTGAN and SMOTE.

1) *CVAE for synthetic data generation*: Table II shows that *Mix* outperforms *Base* across all metrics, for all scenarios of CVAE. The main takeaway from this observation is that CVAE improves the classifier’s performance by adding synthetically generated data to the training set. In the following, we comment the numerical results in greater detail.

F1-score. Comparing the various scenarios of data augmentation with CVAE, we observe that *Balanced* (ensuring all classes are equally represented) achieves the best performance, e.g., improves the classifier’s F1-score by 2.88% in case of *Mix* compared to *Base*.

To illustrate the impact of this improvement, we quantify it in the absolute number of number of correct classifications. For example, in a scenario of 50 microwave links and an alarm report every 15 minutes, there is an upper bound of 72000 samples to be classified daily. Even though it is expected that not every 15-minute window will report an alarm, in practice, such improvement in the F1-score leads to hundreds of additional correct classifications. From an industrial perspective, this translates to significant savings in the maintenance costs.

In contrast, we observe that when adding an excessive number of synthetic data, e.g., *Balanced x 2*, the model will perform worse than *Balanced*. This illustrates that adding synthetic data to the training set does not necessarily improve the generalization performance on the test set. Indeed, while extending a dataset with new real measurements is always

beneficial, there is no a-priori guarantee that adding synthetic data will improve the performance of an ML model.

Let us now focus in Table III on the *per-class* F1-score for the *Base* and the *Mix* models. We consider that the per-class F1-score allows us to quantify the impact of adding synthetic data for each class.

Class-2. Observing the F1-score of Class-2 is particularly important, as in this class we removed 80% of training data points, simulating a scenario of extreme data scarcity. Indeed, data scarcity in Class-2 is reflected in the F1-score of the *Base* model, with an F1-score less than 73%. Adding synthetically generated data significantly improves the F1-score by up to 8%. Comparing the various synthetic data generation strategies, we observe that balancing the dataset so that all classes are equally represented in the training set leads to the best performance.

Similarly, for other classes (*Class-0*, *Class-1*, and *Class-3*), we observe that the *Mix* model outperforms the *Base* model augmenting real data with synthetic data.

We conducted the same analysis in the case of Categorical alarms and concluded that data augmentation improves the F1-score of the *Mix* model by around 3% compared to the *Base* model. For brevity, such results are not included in the paper.

We conclude that balancing the representation of classes in the training dataset, i.e., CVAE in the case of *Balanced*, leads to the best test set performance. Therefore, we compare this scenario to state-of-the-art CTGAN and SMOTE that also re-balance the training dataset, i.e., all classes have the same number of data points.

2) *CVAE vs Baselines*: We compare *Balanced* in the case of CVAE with SMOTE and CTGAN and make the following observations. First, we observe the impact of adding synthetically generated data to the training set by comparing *Mix* and *Base* in the case of SMOTE. Table II shows that *Base* outperforms *Mix* across all metrics, e.g., *Mix* has a better F1-score by almost 2% compared to *Base*. Moreover, looking into the per-class F1-score, we notice that, especially for the underrepresented Class-2, adding synthetically data via SMOTE actually degrades the F1-score by 5% (from 73% to 68%). The takeaway of these numerical evaluations is that adding *synthetic* data that does not truthfully represent the *real* data may lead to performance degradation in the test set. This proves that having more data in the training dataset (in a scenario when real data are augmented with synthetically generated data) does not necessarily improve the classifier’s generalization performance. Finally, comparing CVAE to SMOTE, we observe that CVAE is significantly

TABLE III
PER-CLASS F1-SCORE FOR **BASE** AND **MIX** MODELS FOR CVAE, SMOTE AND CTGAN. FOR CVAE, WE REPORT FOUR CASES OF DATA GENERATION: I) BALANCED, II) BALANCED X2, III) FIXED 150, AND IV) FIXED 400. WE REPORT MEANS AND STANDARD ERRORS OVER 5-FOLD CV.

F1-score	Base	Balanced	Balanced x 2	Fixed 150	Fixed 400	SMOTE	CTGAN
Class-0	96.13 (0.76)	96.32 (0.78)	96.32 (0.96)	95.73 (0.95)	96.20 (0.95)	96.05 (0.92)	95.61 (1.01)
Class-1	93.31 (1.03)	94.12 (0.73)	93.63 (1.21)	93.67 (0.86)	93.79 (1.14)	93.18 (1.09)	93.88 (1.15)
Class-2	72.99 (2.51)	80.95 (2.45)	78.14 (1.82)	78.49 (3.01)	78.69 (3.47)	67.54 (2.79)	75.61 (3.70)
Class-3	90.09 (1.49)	92.64 (0.89)	91.79 (0.82)	92.42 (1.17)	92.85 (1.68)	89.14 (1.25)	90.29 (1.38)

better, as it always improves classifier performance by adding synthetically generated data to the training set.

Second, in the case of CTGAN, we notice that the classifier’s performance when adding synthetic data, i.e., *Mix* scenario, is either similar or slightly better than when only real data are used, i.e., *Base* scenario. In particular, in Table II, we observe an improvement of 0.24% in Accuracy and 0.72% in F1-score. In terms of the per-class F1-score, in Table III, we observe that for Class-2 (being the least represented class), CTGAN ensures an F1-score improvement by 2.62% when the classifier is trained with mixed (real+synthetic) data compared to when the classifier is trained with real data only. For the other classes, the performance is comparable, e.g., CTGAN ensures an improvement by 0.57% in the case of Class-1. We clearly see that CTGAN outperforms SMOTE, as it preserves or improves the classifier’s performance when adding synthetic data to the training set. However, the performance improvement of CTGAN is inferior compared to CVAE, as CVAE improves the F1-score by 8% in the case of Class-2.

VI. CONCLUSION AND FUTURE WORK

Motivated by the high operational costs required for collecting and annotating data, we proposed two Data-Centric approaches for ML-based hardware-failure identification in microwave networks: offline dataset augmentation with CVAE, and online data collection guided by BUAL.

Our main takeaway is that Data-Centric approaches are mandatory for ensuring time and cost-efficient deployment of robust ML models for failure management in microwave networks. Quantitatively, we have demonstrated that BUAL achieves the best-known level of test-set performance with 4.5x fewer training samples, which translates into proportional savings in terms of manual labeling time and waiting time before deployment (three days vs. two weeks). Moreover, we illustrated that synthetically augmenting an offline dataset with CVAE-generated synthetic data improves the F1-score by 2.5% in a scenario characterized by extreme data scarcity (only a few tens of samples for the least-represented class).

We note that for other applications, e.g., when other microwave vendors and operators are considered, our proposed solution might not necessarily work as a ‘*black-box solution*’, and further adjustment might be needed. For instance, in scenarios of data abundance, e.g., over 100k data points, the PCA decomposition and the repeated KMeans clustering in BUAL might not scale very well. Moreover, for synthetic data generation, large datasets would make it feasible to leverage complex CTGANs instead of our CVAE. Nevertheless, we argue that our empirical findings can be practically useful for

many problems in ML for communications networks where deployment costs are dominated by data annotation.

Possible future research directions leveraging this dataset include the development of Online Learning approaches by considering the hardware failure monitoring data as a time series. The goal would be to build real-time processing pipelines for monitoring data and promptly flagging any unknown, anomalous behavior. As a result, even in case of unexpected failure events, the time-to-repair will be kept to a minimum.

Lastly, we plan to make public, for the first time to the best of our knowledge, a high-quality dataset for failure identification in microwave networks collected from a real, currently operating microwave network. We hope that our contribution will open up new research directions in the field of ML for microwave failure management, and will serve as a benchmark dataset for future data-driven failure management frameworks.

REFERENCES

- [1] B. Shariati, M. Ruiz, J. Comellas, and L. Velasco, “Learning from the optical spectrum: Failure detection and identification,” *Journal of Lightwave Technology*, vol. 37, no. 2, pp. 433–440, 2019.
- [2] F. Musumeci, C. Rottondi, A. Nag, I. Macaluso, D. Zibar, M. Ruffini, and M. Tornatore, “An overview on application of machine learning techniques in optical networks,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1383–1408, 2019.
- [3] F. Musumeci *et al.*, “Supervised and semi-supervised learning for failure identification in microwave networks,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1934–1945, 2021.
- [4] D. Zha, K.-H. Lai, F. Yang, N. Zou, H. Gao, and X. Hu, “Data-centric ai: Techniques and future perspectives,” in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 5839–5840.
- [5] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, *A Coding Approach to Event Correlation*. Boston, MA: Springer US, 1995, pp. 266–277.
- [6] H. Wietgreffe *et al.*, “Using neural networks for alarm correlation in cellular phone networks,” in *International Workshop on Applications of Neural Networks to Telecommunications (IWANN)*. Citeseer, 1997, pp. 248–255.
- [7] P. Szilagyi and S. Novaczki, “An automatic detection and diagnosis framework for mobile communication systems,” *IEEE Transactions on Network and Service Management*, vol. 9, no. 2, pp. 184–197, 2012.
- [8] P. Casas, P. Fiadino, and A. D’Alconzo, “Machine-learning based approaches for anomaly detection and classification in cellular networks,” in *8th Traffic Monitoring and Analysis (TMA2016) Workshop*, 04 2016.
- [9] F. Ahmed, J. Erman, Z. Ge, A. X. Liu, J. Wang, and H. Yan, “Detecting and localizing end-to-end performance degradation for cellular data services based on tcp loss ratio and round trip time,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3709–3722, 2017.
- [10] J. Wu, P. P. C. Lee, Q. Li, L. Pan, and J. Zhang, “Cellpad: Detecting performance anomalies in cellular networks via regression analysis,” in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, 2018, pp. 1–9.

- [11] F. Lateano, O. Ayoub, F. Musumeci, and M. Tornatore, "Machine-learning-assisted failure prediction in microwave networks based on equipment alarms," in *2023 19th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2023, pp. 1–7.
- [12] O. Ayoub *et al.*, "Explainable artificial intelligence in communication networks: A use case for failure identification in microwave networks," *Computer Networks*, vol. 219, p. 109466, 2022.
- [13] O. Ayoub, F. Musumeci, F. Ezzeddine, C. Passera, and M. Tornatore, "On using explainable artificial intelligence for failure identification in microwave networks," in *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2022, pp. 48–55.
- [14] L. Pan, J. Zhang, P. P. Lee, M. Kalander, J. Ye, and P. Wang, "Proactive microwave link anomaly detection in cellular data networks," *Computer Networks*, vol. 167, p. 106969, 2020.
- [15] A. Shahraki, M. Abbasi, A. Taherkordi, and A. D. Jurcut, "Active learning for network traffic classification: A technical study," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 1, pp. 422–439, 2022.
- [16] Q. Xu and R. Zheng, "When data acquisition meets data analytics: A distributed active learning framework for optimal budgeted mobile crowdsensing," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.
- [17] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.
- [18] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," in *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021.
- [19] Y. Gal, R. Islam, and Z. Ghahramani, "Deep bayesian active learning with image data," in *International conference on machine learning*. PMLR, 2017, pp. 1183–1192.
- [20] V.-L. Nguyen, M. H. Shaker, and E. Hüllermeier, "How to measure uncertainty in uncertainty sampling for active learning," *Machine Learning*, vol. 111, no. 1, pp. 89–122, 2022.
- [21] E. Ayanoglu, K. Davaslioglu, and Y. E. Sagduyu, "Machine learning in nextg networks via generative adversarial networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 2, pp. 480–501, 2022.
- [22] H. Navidan, P. F. Moshiri, M. Nabati, R. Shahbazian, S. A. Ghorashi, V. Shah-Mansouri, and D. Windridge, "Generative adversarial networks (gans) in networking: A comprehensive survey & evaluation," *Computer Networks*, vol. 194, p. 108149, 2021.
- [23] Y. Yang, Y. Li, W. Zhang, F. Qin, P. Zhu, and C.-X. Wang, "Generative-adversarial-network-based wireless channel modeling: Challenges and opportunities," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 22–27, 2019.
- [24] I. William H Clark, S. Hauser, W. C. Headley, and A. J. Michaels, "Training data augmentation for deep learning radio frequency systems," *The Journal of Defense Modeling and Simulation*, vol. 18, no. 3, pp. 217–237, 2021.
- [25] K. Davaslioglu and Y. E. Sagduyu, "Generative adversarial learning for spectrum sensing," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.
- [26] B. Tang, Y. Tu, Z. Zhang, and Y. Lin, "Digital signal modulation classification with data augmentation using generative adversarial nets in cognitive radio networks," *IEEE Access*, vol. 6, pp. 15 713–15 722, 2018.
- [27] M. Massaoudi, S. S. Refaat, and H. Abu-Rub, "Intrusion detection method based on smote transformation for smart grid cybersecurity," in *2022 3rd International Conference on Smart Grid and Renewable Energy (SGRE)*, 2022, pp. 1–6.
- [28] M. Sun, H. Qian, K. Zhu, D. Guan, and R. Wang, "Ensemble learning and smote based fault diagnosis system in self-organizing cellular networks," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–6.
- [29] S. A. Abdulkareem, C. H. Foh, F. Carrez, and K. Moessner, "Smote-stack for network intrusion detection in an iot environment," in *2022 IEEE Symposium on Computers and Communications (ISCC)*, 2022, pp. 1–6.
- [30] A. Tesfahun and D. L. Bhaskari, "Intrusion detection using random forests classifier with smote and feature reduction," in *2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies*, 2013, pp. 127–132.
- [31] M. Razghandi, H. Zhou, M. Erol-Kantarci, and D. Turgut, "Variational autoencoder generative adversarial network for synthetic data generation in smart home," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 4781–4786.
- [32] Y. Qu, H. Ma, Y. Jiang, L. Wang, and J. Yu, "A network data reinforcement method based on the multiclass variational autoencoder," *Security and Communication Networks*, vol. 2022, pp. 1–10, 07 2022.
- [33] L. Z. Khan, J. Pedro, N. Costa, L. De Marinis, A. Napoli, and N. Sambo, "Data augmentation to improve performance of neural networks for failure management in optical networks," *Journal of Optical Communications and Networking*, vol. 15, no. 1, pp. 57–67, 2023.
- [34] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," *Advances in neural information processing systems*, vol. 32, 2019.
- [35] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.
- [36] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," *Advances in neural information processing systems*, vol. 32, 2019.
- [37] D. Elreedy and A. F. Atiya, "A comprehensive analysis of synthetic minority oversampling technique (smote) for handling class imbalance," *Information Sciences*, vol. 505, pp. 32–64, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0020025519306838>
- [38] A. Malinin, L. Prokhorenkova, and A. Ustimenko, "Uncertainty in gradient boosting via ensembles," in *International Conference on Learning Representations*, 2021.
- [39] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.
- [40] G. Citovsky *et al.*, "Batch active learning at scale," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021.
- [41] J. T. Ash, C. Zhang, A. Krishnamurthy, J. Langford, and A. Agarwal, "Deep batch active learning by diverse, uncertain gradient lower bounds," in *International Conference on Learning Representations*, 2020.
- [42] K. Pearson, "On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, vol. 2, no. 11, pp. 559–572, 1901.
- [43] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.
- [44] A. Pagnoni, K. Liu, and S. Li, "Conditional variational autoencoder for neural machine translation," *arXiv preprint arXiv:1812.04405*, 2018.
- [45] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [46] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794.

4.2. As-Soon-As-Possible Hardware Fault Classification with Guarantees

Though ML for fault management is a promising technology, it is difficult to convince network operators to deploy it in the field. This is because of the black-box nature of ML models, which makes their underlying decision-making inscrutable by humans. In general, even though we may extensively test ML models and ascertain their performance on benchmark datasets, we do not have any guarantee of the reliability of individual predictions at deployment time. This paper considers this very problem in the context of fault management in microwave networks. Specifically, we consider the problem of training *probabilistic* ML fault classifiers, i.e., models that not only provide a prediction, but also a probability of correct classification. This allows network operators to quantitatively assess the risk in accepting a ML model's predictions, and thus make informed decisions. However, training probabilistic ML models is a difficult task. Indeed, though the most popular ML class output normalized scores for each output, these scores do not map well to probabilities. Indeed, state-of-the-art ML models tend to be overconfident in their predictions (i.e., their predicted scores are higher than the actual probability of correct classification). This can have potentially catastrophic consequences in fault management, as overconfident models instill a false sense of confidence in their users. We propose solving this problem by leveraging recent results in the field of Conformal Prediction, namely, Venn-Abers Predictors (VAPs), for training probabilistic classifier. The fundamental idea of VAPs is to perform a post-hoc calibration step of a trained ML model, such that its predicted scores are matched with the statistical distribution of the data. We apply VAPs to the task of ASAP classification of hardware faults. In particular, we assume that telemetry data is being streamed to the ML model, which will return a prediction only if the probability of correct classification exceeds a user-specified threshold. Our numerical results on a real-world dataset illustrate the capability of VAPs to satisfy tight constraints on the probability of correct classification (namely, 95% and 99%) while reducing the time-to-predict by several times compared to a state-of-the-art approach based on fixed-size telemetry windows. Overall, this work raises the fundamental issue of reliability in ML applied to fault management, and proposes a working solution towards in-production fully-reliable ML fault management systems. Also for this work, we made our dataset, comprising hours of manually-annotated telemetry data from real-world microwave networks, publicly available to the community.

ASAP Hardware Failure-Cause Identification in Microwave Networks using Venn-Abers Predictors

Nicola Di Cicco, *Graduate Student Member, IEEE*, Memedhe Ibrahim, *Member, IEEE*
 Omran Ayoub, *Member, IEEE*, Federica Bruschetta, Michele Milano,
 Claudio Passera, and Francesco Musumeci, *Senior Member, IEEE*

Abstract—We investigate classifying hardware failures in microwave networks via Machine Learning (ML). Although ML-based approaches excel in this task, they usually provide only hard failure predictions without guarantees on their reliability, i.e., on the probability of correct classification. Generally, accumulating data for longer time horizons increases the model’s predictive accuracy. Therefore, in real-world applications, a trade-off arises between two contrasting objectives: i) ensuring high reliability for each classified observation, and ii) collecting the minimal amount of data to provide a reliable prediction. To address this problem, we formulate hardware failure-cause identification as an *As-Soon-As-Possible (ASAP) selective classification problem* where data streams are sequentially provided to an ML classifier, which outputs a prediction as soon as the probability of correct classification exceeds a user-specified threshold. To this end, we leverage Inductive and Cross Venn-Abers Predictors to transform heuristic probability estimates from any ML model into rigorous predictive probabilities. Numerical results on a real-world dataset show that our ASAP framework reduces the time-to-predict by $\sim 8x$ compared to the state-of-the-art, while ensuring a selective classification accuracy greater than 95%. The dataset utilized in this study is publicly available, aiming to facilitate future investigations in failure management for microwave networks.

Index Terms—Microwave networks, failure-cause identification, As-Soon-As-Possible classification, Venn-Abers predictors

I. INTRODUCTION

MICROWAVE networks are widely deployed as an alternative technological solution to optical backbones, especially to support backhauling of mobile traffic. Next-generation (6G) communication services supported by such networks are characterized by extreme availability requirements such as six 9s or even higher reliability [1]–[3]. Therefore, prompt failure management represents a key factor for the success of microwave networks in the 6G ecosystem. A quick and reliable hardware failure-cause identification, as well as precise discrimination of faulty devices, are of paramount importance, as the countermeasures adopted by network operators to address network failures (e.g., whether to reconfigure, repair, or even substitute a network device) strongly depend on these two factors. In this context, early hardware failure

N. Di Cicco and M. Ibrahim are co-first authors of this paper. N. Di Cicco, M. Ibrahim, and F. Musumeci are with the Department of Electronics, Information, and Bioengineering (DEIB), Politecnico Di Milano, Italy. E-mail: {name}. {surname}@polimi.it. O. Ayoub is with the Department of Innovative Technologies (DTI), University of Applied Sciences of Southern Switzerland, Viganello, Switzerland. E-mail: omran.ayoub@supsi.ch. F. Bruschetta, M. Milano and C. Passera are with SIAE Microelettronica S.p.A., Italy.

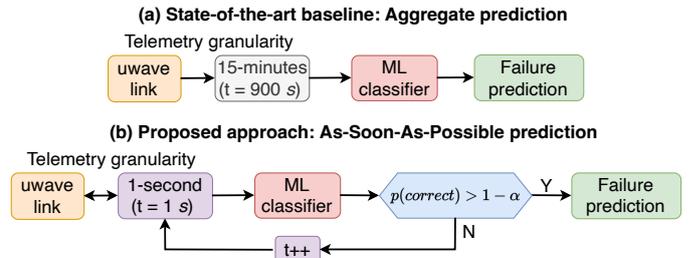


Fig. 1. (a) Aggregate prediction vs. (b) As-Soon-As-Possible prediction. Aggregate prediction collects network telemetry for a fixed-size observation window, and then feeds it to an ML-based failure classifier. In contrast, As-Soon-As-Possible prediction returns a failure prediction as soon as the probability of correct classification exceeds a user-specified threshold $1 - \alpha$.

detection and cause identification not only helps in lowering mean-time-to-repair (MTTR) but also allows effective service maintenance at a higher level, e.g., achieved by rerouting traffic from a malfunctioning link towards an operating one while link troubleshooting and repairing operations take place.

Current hardware failure classification approaches are based on observing network parameters (e.g., transmitted/received power measures, configured modulation format, and other links settings) and equipment alarms retrieved by a Network Management System (NMS). To output a failure diagnosis and hence devise an appropriate mitigation strategy, operators rely on knowledgeable domain experts, who typically observe and analyze the retrieved information case by case. Due to the increased complexity of modern microwave networks and the large volumes of data that need to be analyzed by domain experts to extract valuable information on failure causes, there is an urgent need to automate the failure management process. To this end, Artificial Intelligence (AI) and Machine Learning (ML) are key enablers, as shown by the large amounts of studies that recently appeared in literature [4]–[8].

However, although AI/ML models achieve satisfactory predictive performance in hardware failure-cause classification, obtaining a reliable model that provides an uncertainty measure of its outputs is crucial for real deployments, as it allows operators to make informed decisions on how to handle failures properly. To reach this objective, we leverage Uncertainty Quantification (UQ) in ML models, specifically Inductive and Cross Venn-Abers predictors [9], a family of probabilistic predictors with formal validity guarantees closely related to the field of Conformal Prediction [10]. We leverage Venn-Abers predictors to develop ML-based hardware failure classifiers that output not only the most likely root cause of a given hardware failure but also the probability that

the prediction is correct. This additional information on the predictive uncertainty can be exploited by domain experts to properly gauge the risk of making decisions based on the ML model's outputs.

Fig. 1 shows a state-of-the-art solution based on *Aggregate prediction* and our proposed solution based on *As-Soon-As-Possible prediction*, aiming to make a prediction as-soon-as-possible, once the required statistical guarantees are met. Prior works that perform Aggregate prediction [11], [12] collect fixed-size 15-minute windows of equipment alarms and feed the data to an ML classifier for hardware failure-cause prediction. Conversely, *As-Soon-As-Possible* prediction considers *one-second* telemetry granularity of equipment alarms, and feeds the data to an ML classifier that makes a failure-cause classification only if a statistical guarantee is met. In case not, new data from the next observation second is queried. The process is repeated until the statistical guarantee is met.

Intuitively, a reliable prediction, i.e., failure-cause classification with low uncertainty, can be obtained after a sufficient amount of information (e.g., a data stream including information on the status of multiple equipment alarms) has been collected and fed to the AI/ML model. However, this is in contrast with the ideal goal of *As-Soon-As-Possible* (ASAP) prediction, where an operator wishes to react to failures as soon as they occur or, more realistically, with a minimal amount of information (and hence, after a minimal amount of time) sufficient to obtain a reliable prediction. As a motivating example, we quantify the performance gap between classifying a hardware failure at the first second and classifying at the end of a 15-minute window on our alarms dataset (described in detail in Section III). Considering a state-of-the-art XGBoost [13] model, failure classification at the first second and at the end of the 15-minute window result in cross-validated accuracy of $88\% \pm 2\%$ and $96\% \pm 2\%$, respectively. We conclude that, though the first-second classification already yields a very good performance, the gap with the 15-minute classification is practically significant. In particular, in the context of failure management, it is paramount to achieve near-perfect failure classification, as predicting the wrong failure cause may result in choosing inappropriate or dangerous mitigation strategies.

Therefore, the research question we aim to address in this paper is: *Can we autonomously return a maximally accurate hardware failure-cause prediction in the least amount of time?*

To solve the above problem, we propose an ML-based hardware failure-cause classification framework that returns a prediction as soon as the probability of correct classification is greater or equal to a user-specified threshold (e.g., 95% or 99%). In other words, our proposed framework can provide failure-cause predictions that are both *timely* and *highly reliable*. Compared to our prior work [14], the main novelties proposed in this paper can be summarized as follows: 1) we explicitly address a reliability aspect of probabilistic prediction, while our prior work only considered hard predictions, and 2) we redefine the problem by considering ASAP failure-cause classification at one-second granularity, while in the prior work, we considered fixed-size observation windows to forecast alarm states and corresponding failure causes without any statistical guarantee. Our key contributions

are summarized as follows:

- We introduce a new dataset for ML-based failure management, comprising alarms and ground-truth annotations indicating failure causes in microwave links from a real-world microwave network, and make it publicly available to the research community.¹ (Section III)
- We propose a principled methodology for ML-based *As-Soon-As-Possible* hardware failure-cause classification in microwave networks, such that the ML model retrieves from the network the minimal amount of information on device alarms to output a prediction only when its probability of correct classification is at least above a user-specified safety threshold. To achieve this, we leverage Venn-Abers predictors, which offer theoretical guarantees on predictive probabilities. (Section IV)
- We validate our approach against the current state-of-the-art, illustrating that our approach consistently yields better probabilistic predictions, thereby allowing for reliable selective classification. Furthermore, we explore and discuss tunable performance trade-offs introduced by our proposed methodology. (Section V)

II. RELATED WORK

The application of ML for failure detection and failure-cause identification in telecommunication networks is receiving considerable attention, as it offers operators the ability to take mitigation actions promptly [15]–[17]. In this section, we discuss some recent literature utilizing ML for failure management with a specific focus on microwave networks.

Prior works utilized ML for detecting failures due to transmission parameter degradation or attenuation on microwave links [18]–[20]. For instance, in [18], authors propose an ML-based approach for continuously monitoring the performance of a microwave link and detecting degradation due to natural weather conditions, leveraging on performance measurements, such as signal strength and signal-to-noise ratio, and weather information. Another work [19] proposes an approach based on Long Short-Term Memory (LSTM) and recurrent neural networks to continuously predict rain-induced attenuation due to weather conditions using past measurements. Similarly, authors in [20] propose various ML-based approaches for real-time analysis of the link's performance and forecasting of rain-induced attenuation leveraging historical data.

Other works have utilized data available from microwave links to predict a broader set of failures. For instance, [21] proposes supervised and semi-supervised learning approaches for failure-cause identification leveraging link performance data from a nationwide microwave network. Gathered data measurements were aggregated in 15-minute intervals, and ML techniques were employed to train models to identify six categories of failure causes, achieving classification accuracy up to 95%. In [22], authors focus on tackling the same problem considering, in addition to the link's performance measurements, alarm data stemming from devices and data relative to weather and terrain surrounding the microwave link. Authors devise a deep learning-based method that achieves a 95%

¹<https://github.com/bonsai-lab-polimi/tmsm2024-asap-venn-abers>

classification accuracy. Authors in [23] introduce an anomaly detection system that leverages both the link's performance data (signaling quality and transmission performance) and network topology, while employing active learning techniques to update the detection model continually. The performance data is aggregated and sent to a network management system every 15 minutes, where it is later processed and used for inference. Similarly, in [11], authors introduce an LSTM-based feature fusion network designed to capture both spatial and temporal features within microwave network by incorporating network topology into the LSTM, in addition to links' performance data collected by probes every 15 minutes. Moreover, [12] proposes an ML-based framework that combines eXplainable AI techniques and uncertainty quantification to achieve reliable and robust failure-cause identification. As data, the work utilizes the link's performance measurements aggregated in 15-minute intervals. The work in [24] also examines the problem of failure-cause identification in a scenario involving multiple cooperating operators, i.e., where data is split among operators, and where one operator possesses only partial knowledge of failure causes during the training stage. The authors devised a classification model based on Federated Learning (FL) to train an ML model to detect six distinct failure causes while adhering to privacy constraints.

While these works rely on accumulating telemetry statistics to subsequently perform failure-cause detection, our proposed framework aims to bridge the gap between prediction and telemetry accumulation, where predictions and telemetry accumulation occur concurrently, allowing the ML models to output predictions (in our case, indicating the hardware failure-cause in a microwave link) as soon as they reach a mature stage, referred to as *As-Soon-As-Possible* prediction. By intertwining ML model inference and performance assessment with ongoing telemetry and data collection, our proposed frameworks strive to offer more robust, timely, and actionable failure identification and mitigation.

Finally, the methodological framework proposed in this paper closely resembles ASAP in-network traffic classification in pForest [25], from which we adopt the terminology. Our work brings two major improvements compared to pForest. First, we propose integrating Venn-Abers calibration in the inference phase to provide formal guarantees on the probability of correct classification, while pForest does not provide any such guarantee. Second, we do not make any assumption on the underlying ML model, namely, we do not assume a specific model architecture, nor a minimum level of performance (e.g., classification accuracy) in generalization, while pForest is limited to Random Forests and assumes that the model can deliver a minimum level of performance. Our goal is, therefore, to provide the network manager with an accurate decision-making tool in the form of valid predictive probabilities, regardless of how good or bad the underlying ML model might be.

III. BACKGROUND ON MICROWAVE NETWORKS

In this Section, we provide a brief overview of the main components of a microwave network and introduce our dataset of hardware failures in microwave networks.

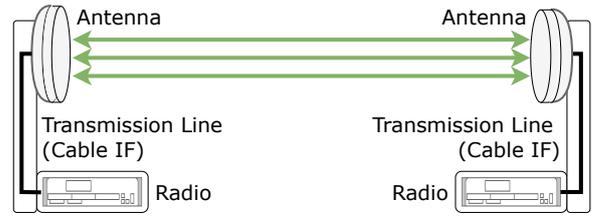


Fig. 2. Basic components of a microwave link.

TIMESTAMP	A_1	A_2	A_3	...	A_{164}	$\sum(A_1)$	$\sum(A_2)$	$\sum(A_3)$...	$\sum(A_{164})$	label
29-03-2023 10:00:01	1	0	1	...	1	1	0	1	...	1	IDU
29-03-2023 10:00:02	1	1	0	...	0	2	1	0	...	1	IDU
29-03-2023 10:00:03	1	0	0	...	0	3	0	0	...	1	IDU
29-03-2023
29-03-2023 10:14:59	1	0	1	...	0	899	32	2	...	1	IDU
29-03-2023 10:15:00	1	1	0	...	0	900	33	2	...	1	IDU
29-03-2023
29-03-2023 17:15:01	0	0	1	...	1	0	0	1	...	1	ODU
29-03-2023 17:15:02	1	0	1	...	0	1	0	2	...	1	ODU
29-03-2023
29-03-2023 17:30:00	0	0	1	...	1	45	0	900	...	756	ODU

Fig. 3. Illustrative representation of the SIAE-Microelettronica hardware failure dataset for one microwave link in two 15-minute windows.

A. Microwave link

Fig. 2 shows the basic structure of a bidirectional microwave link, highlighting the *transmitting end* (TX/RX site) and the *receiving end* (RX/TX site). Each end is composed of three main elements:

- 1) *Microwave Radio*: generates a signal at the TX/RX site and receives the signal at the RX/TX site. In this work, we consider a split-mount placement, where the electronic devices are distributed between an outdoor unit (ODU) and an indoor unit (IDU). The IDU contains the power unit and electronic components like modems, converters, and the system's interface. The power unit in the IDU provides the necessary electrical power to the entire system, including the ODU. The ODU is mounted outside from the antenna and is powered via the cable from the IDU.
- 2) *Transmission Line*: connects the microwave radio to the directional antenna through a coaxial cable or a waveguide. The Intermediate Frequency cable (IF cable) connects the IDU and the ODU in a split-mount system. The IF cable is responsible for non-negligible signal losses, depending on signal frequency, and may strongly affect the quality of transmission in case of physical medium deterioration.
- 3) *Microwave Antenna*: is directional, usually parabolic-shaped, and characterized by its gain, size, and directivity functions. In split-mount systems, the antenna is co-located with the ODU.

We focus on hardware failures that can impact microwave equipment of a microwave link.² In particular, 1) hardware failure of the IDU unit (IDU failure), 2) hardware failure of the ODU unit (ODU failure), 3) hardware failure of the

²From the three components of the microwave link, we consider hardware failures of the microwave radio and the transmission line. However, we do not consider hardware failures of the antenna.

transmission line (IF cable failure), and 4) hardware failure of the power unit (power failure). In the following, we provide a detailed description of the real-world hardware failure dataset used in our study.

B. Microwave hardware failure bit-sequence dataset

The unavailability of a microwave link is defined in ITU-T Recommendations G.826 and G.828 [26] in terms of *Unavailability Seconds* (UAS), i.e., the number of seconds over which the number of errored bits exceeds a given threshold. UAS may be caused by several phenomena, such as propagation failures due to e.g., atmospheric fading, or hardware failures due to equipment malfunction.

We leverage a real-world dataset of hardware failures from 108 microwave links from a microwave network provided by SIAE Microelettronica [27]. Alarms from each microwave link are collected at *one second* granularity in windows of 15 minutes, with a binary indicator “1” if an alarm signal is ON and a binary indicator “0” if an alarm signal is OFF. Hence, constructing the *alarm bit-sequence dataset*. In addition, from the *alarm bit sequence dataset*, we construct a *15-minute window dataset* by computing the number of seconds an alarm signal is ON in non-overlapping 15-minute windows. For each alarm signal, in a 15-minute window, we get a number ranging between 0 and 900, representing the number of seconds the alarm signal was ON during the 15-minute window.³

Our dataset corresponds to 861 disjoint 15-minute window observations of 164 alarm signals collected from 108 microwave links during a time period across two years. Alarm signals are triggered based on the output of multiple telemetry sources installed in the hardware equipment, such as temperature and power sensors, or status monitors for multiple hardware and software sub-components.

The dataset comprises four hardware failure classes: 1) *IDU failure* (e.g., failure of some electronic IDU component, or a temperature issue due to improper equipment installation and/or a worn fan), 2) *ODU failure* (similar to IDU), 3) *Cable failure* (e.g., damaged connectors), and 4) *Power failure* (e.g., due a power outage and/or a battery problem). Each failure type is identified based on alarms issued by the radio equipment, serving as input features to the ML-based classifier. The frequency of each failure class in each 15-minute window is as follows: 1) IDU failure: 129 observations, 2) ODU failure: 493 observations, 3) Cable failure: 75 observations, 4) Power failure: 164 observations.

Fig. 3 shows an illustrative example of the hardware failure dataset for one microwave link. For each link, we consider a *one-second* granularity timestamp observation for each of the 164 alarm signals (A_1 to A_{164}) in 15-minute non-overlapping windows. Additionally, we report the cumulative sum of the number of seconds an alarm signal is ON during the 15-minute

³A value “0” means the alarm signal is OFF during the whole window, while a value “900” means the alarm signal is ON during the whole window duration. Any number x : $0 < x < 900$, means the alarm signal is ON for x seconds during the whole window duration. However, this does not imply the alarm signal is ON for x seconds sequentially, as, in practice, an alarm signal may be ON and OFF in different sections of the 15-minute window.

window, for each alarm signal ($\sum(A_1)$ to $\sum(A_{164})$). Finally, we report the ground truth *label* of the hardware failure-cause.

We first reconstruct the *bit-sequence dataset* measured every second within the 15-minute telemetry collection window. Then, we construct expanding-window features starting from the beginning of each 15-minute window. Each expanding-window feature represents the number of seconds the corresponding alarm signal was ON at time $t = 1, \dots, 900$ within the 15-minute window. We leverage this dataset to simulate a streaming scenario where telemetry data is aggregated and fed to an ML model for failure-cause classification in real-time. Note that the 15-minute dataset is a subset of this new dataset. In summary, two datasets are used in our study are:

- 1) *Expanding window dataset* for the As-Soon-As-Possible prediction. Each observation represents the total number of seconds the alarm signals were ON at a certain time within the 15-minute observation window.
- 2) *15-minute window dataset* for Aggregate prediction. Each observation aggregates bit sequences in non-overlapping windows of 15 minutes, counting for each window the number of seconds each alarm signal is ON.

IV. AS-SOON-AS-POSSIBLE FAILURE-CAUSE IDENTIFICATION IN MICROWAVE NETWORKS

We now focus on the problem of failure-cause classification through an ML model, given a stream of microwave equipment alarms. While in this paper we utilize a microwave hardware failure dataset, our methodology can be applied to any dataset of streamed measurements, such as the ones employed in prior work on classifying propagation failures [12]. We first present our proposed As-Soon-As-Possible classification framework, and then discuss its methodological aspects in detail.

A. Reference scenario and problem statement

Our reference scenario is as follows: when a non-zero UAS is detected in a radio link, the alarm bit sequences are streamed to an ML model for inference. In particular, as the stream progresses, we accumulate statistics on the alarm status to construct more accurate representations of the alarm. In our case, we consider computing the total number of seconds each alarm signal is ON. The bit sequences are then fed to an ML model trained for failure-cause classification.

Fig. 4 illustrates and compares a high-level overview of two ML-based methodologies for failure-cause classification, namely, (a) Aggregate prediction, which is the current state-of-the-art, and (b) As-Soon-As-Possible prediction.

Aggregate prediction (Fig. 4, top) consists of accumulating telemetry data for a fixed-size window (in our case, 15 minutes). For failure management in microwave networks, the above methodology displayed remarkable accuracy (above 95% in past literature [21]). However, there are two main drawbacks to this approach. First, aggregate failure classification always requires 15-minute windows, irrespective of the “difficulty” in classifying an observation. Since choosing a mitigation strategy depends on proper failure-cause classification, the aggregate window strategy introduces an unnecessary bottleneck. Second, and more critical, there are

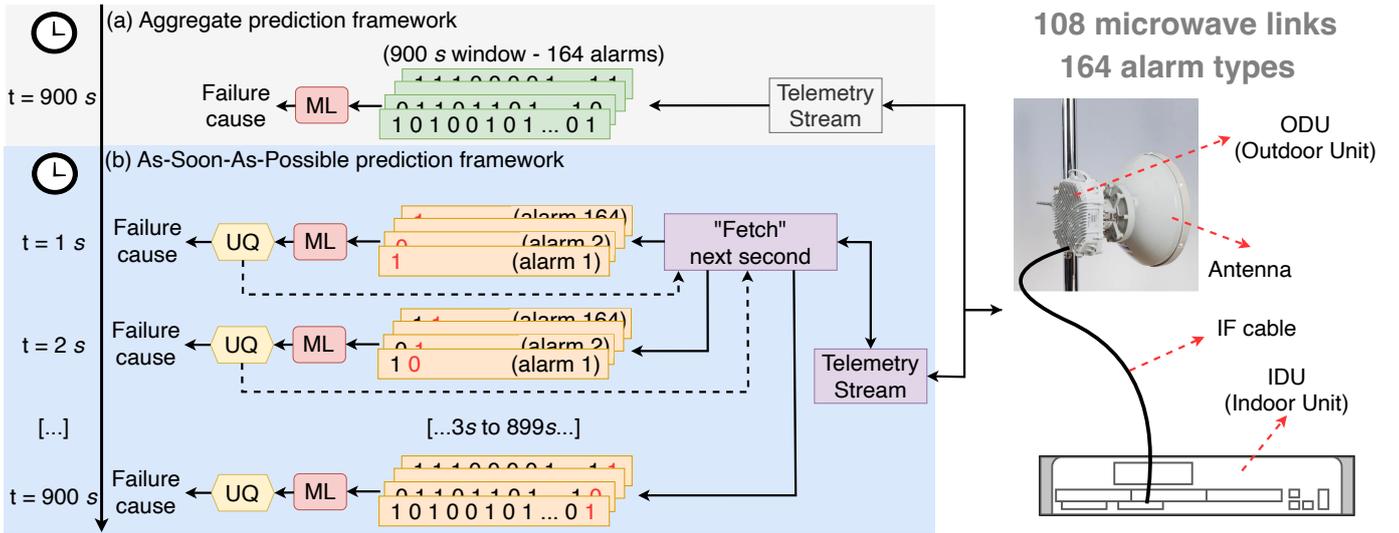


Fig. 4. Aggregate vs. As-Soon-As-Possible (ASAP) failure-cause identification in microwave networks. The aggregate approach collects telemetry data for a fixed-size temporal window (e.g., 15 minutes) and then feeds the extracted features to an ML model for classification. ASAP prediction feeds to the ML model expanding subwindows of telemetry data as soon as they are collected and leverages Uncertainty Quantification (UQ) to return a prediction only when the probability of correct classification is greater than a user-specified safety threshold.

no formal guarantees of the correctness of the prediction. Even if the model displays a test accuracy of 95% (i.e., it performs generally correct class assignment), we have no general guarantees on the probability of correct predictions on new individual samples. In other words, the aggregate method does not quantify how *reliable* the model's predictions are.

To solve the above problems, we propose leveraging *As-Soon-As-Possible* (ASAP) prediction (Fig. 4, bottom). In the ASAP prediction framework, alarms are fed to the ML model as soon as they are collected. By leveraging principled uncertainty quantification via Venn-Abers predictors, the ML classifier will produce a prediction as soon as the probability of correct classification is greater or equal to a user-specified threshold. The advantages of ASAP prediction over aggregate prediction lie in reducing the average time-to-predict (hence, the time for deploying the appropriate mitigation strategy) without compromising on the predictive accuracy. Moreover, by providing a soft probability instead of a hard class assignment, we allow the network operator to make informed decisions based on predictive uncertainty.

Referring to Fig. 4 (b), *As-Soon-As-Possible prediction* shows an example of accumulating the number of seconds each alarm is ON to make a classification decision with high confidence. At a timestamp t (e.g., $t = 2s$), the ML classifier utilizes the information from $t-1$ previous timestamps (e.g., $t = 1s$). In line with this, we aim to deploy a model that returns accurate predictions as soon as possible. This problem can be formally stated as *selective classification*: the model should return a prediction only if the probability of correct classification is greater or equal to a user-specified safety threshold $1 - \alpha$; otherwise, it will abstain. In other words, the classifier should differentiate between easy-to-classify and hard-to-classify hardware failures and decide accordingly. For example, predictions for easy-to-classify failures may be returned at the first second ($t = 1s$). In contrast, for hard-to-classify failures, the model might abstain from predicting until

several hundreds of seconds of alarms have been observed (e.g., $t = 900s$ in case of the complete 15-minute window).

We remark that, when a user enforces a safety threshold (e.g., 95% probability of correct classification), the model may not guarantee that level of certainty for every example at the end of the 900s monitoring window. In other words, in the case of particularly hard-to-classify examples, the model will abstain from predicting. We refer to these samples as *rejected* samples. Rejected samples might be, for instance, sent to domain experts for a more detailed inspection [12]. The rejection rate ultimately depends on the predictive power of the ML model: a highly accurate model implies lower rejection rates, and vice-versa. In this context, we emphasize that our framework enforces safety constraints without assumptions about the underlying ML model's performance.

In the following discussion, we assume that the ML classifier is a *scoring classifier* that can output a heuristic measure of confidence on its predictions, e.g., in the form of normalized scores in $[0, 1]$ for each output class. Many popular ML models are scoring classifiers: for instance, artificial neural networks output softmax probabilities, while decision trees output the class frequency in the leaf nodes. From this assumption, we discuss two different strategies for implementing ASAP failure prediction. First, we discuss the baseline strategy of thresholding the class scores, which is a straightforward extension of the current state-of-the-art, and we highlight its fundamental limitations. We then propose leveraging Venn-Abers predictors to overcome these limitations.

B. ASAP failure classification via score thresholding

A straightforward solution for ASAP failure classification is to threshold the predicted class scores, returning a prediction only if the score associated with the most likely class is greater than $1 - \alpha$. Formally, at each time-instant $t \in [1, 900]$ in the alarm collection window, we return a class prediction \hat{y}_t given features \mathbf{x}_t as follows:

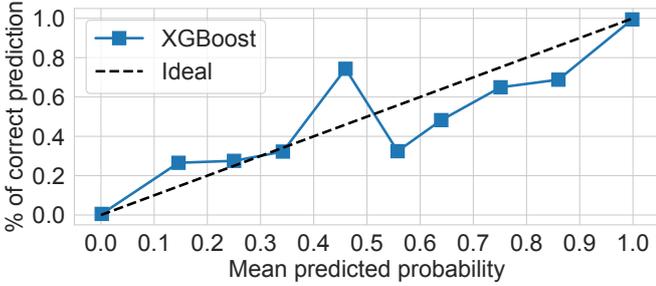


Fig. 5. Reliability diagram of an XGBoost model on the hardware failures dataset. The XGBoost model is overconfident for high-confidence predictions.

$$\hat{y}_t = \begin{cases} \arg \max \hat{f}(\mathbf{x}_t) & \text{if } \max \hat{f}(\mathbf{x}_t) \geq 1 - \alpha, \\ \emptyset & \text{otherwise,} \end{cases} \quad (1)$$

where \emptyset stands for “no prediction”. Unfortunately, the classification scores produced by an ML model are generally not valid probabilities in the statistical sense [28], [29]. Formally, we say that a model is *calibrated* if [29]:

$$p(c_j | p^{c_j}) = p^{c_j} \quad \forall p^{c_j} \in [0, 1], \quad (2)$$

where p^{c_j} is the predicted probability for class c_j , and $p(c_j | p^{c_j})$ is the probability of the ground-truth being c_j if the corresponding predicted probability is p^{c_j} . For example, in a perfectly calibrated model, a class prediction with a score equal to 0.95 has a 95% chance of being correct. Generally, a trained ML model is not calibrated out of the box [28], [29]. As an illustrative example, Fig. 5 shows the *reliability diagram* of an XGBoost classifier in an 80-20 train-test split of our 15-minute window dataset. This diagram displays the mean predicted probability versus the empirical frequency of correct classification in ten equally-spaced bins between $[0, 1]$. An ideal, perfectly calibrated model would display points located on the diagonal of the diagram. In our case, we observe that the XGBoost model is, on average, not well-calibrated. For example, for a mean predicted probability of 86%, the actual frequency of correct classifications is only 69%. We conclude that, though the model is very skilled at *class assignment* ($> 95\%$ test set accuracy), it is *overconfident* when outputting high-confidence predictions. In other words, the high predictive power of the current state-of-the-art does not guarantee probability calibration, which is highly undesirable for high-risk failure management applications. Though this illustrative example considers one specific split and model class, in general, we have no control on the default calibration of an ML model. We, therefore, need more sophisticated methodologies for producing reliable, high-confidence predictions that do not assume the model to be calibrated by default.

We consider the problem of *probability calibration*, that is, how to transform heuristic uncertainty estimates of an ML model into valid probabilities. Popular classical approaches for probability calibration are Platt scaling [30] and isotonic regression [31]. Unfortunately, the correctness of these approaches depends on assumptions that are unrealistic in many practical scenarios. In particular, Platt scaling assumes that the reliability diagram of the ML classifier to be calibrated has a sigmoidal shape, while isotonic regression assumes a perfectly

monotonic relationship between the predicted scores and the true probabilities. In summary, classical approaches like Platt scaling and isotonic regression can be regarded as heuristics with no formal validity guarantees.

C. ASAP failure classification via Venn-Abers predictors

An alternative approach emerging as a promising solution to overcome the shortcomings of the aforementioned approaches are Venn-Abers predictors (VAPs) [9], [10]. VAPs are a methodological framework for turning heuristic probability estimates from an arbitrary ML model into rigorous probability estimates with theoretical guarantees on calibration. The only assumption is the availability of a *calibration dataset* $\mathcal{D}_{\text{cal}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_{\text{cal}}}$, where \mathbf{x} and y indicate features and ground-truth annotations, respectively. The calibration set represents “fresh” i.i.d. data not used during training. VAPs leverage the ML model’s predictive performance on the calibration set to turn class scores into well-calibrated probabilities. VAPs are, in principle, defined for binary classification problems. We first briefly discuss the algorithm for the binary classification case and then extend it to the multiclass case.

Binary VAPs. We consider two variants of VAPs, namely Inductive Venn-Abers predictors (IVAPs) and Cross Venn-Abers (CVAPs) predictors. IVAPs provide formal theoretical guarantees on calibration. CVAPs are derived from IVAPs, exhibiting stronger empirical performance on average while dropping theoretical guarantees on calibration.

It can be demonstrated that, unfortunately, it is impossible to learn an optimal probabilistic classifier from a finite-size dataset [32], [33]. VAPs overcome this challenge by a) producing two probabilities instead of one, and b) restricting the optimality guarantees to calibration.

Formally, VAPs are *multiprobabilistic predictors*, that is, for each test example, they produce two probabilities (p_0, p_1) for the sample to be assigned to the positive class. VAPs guarantee that either p_0 or p_1 will be perfectly calibrated, according to the definition in Eq. (2). The key intuition is as follows: if p_0 and p_1 are close to each other, and one of them is calibrated, then we can expect that their “average” will be also calibrated.

Indeed, for decision-making purposes, we need one single probability value. Unfortunately, it is generally impossible to know which one among p_0 or p_1 is the “right” choice. To solve this problem, p_0 and p_1 are merged into a single value p that minimizes the error with respect to a proper scoring rule, e.g., the log-loss or the Brier score. We now discuss each of the above steps in more detail.

Algorithm 1 and Algorithm 2 illustrate the procedure for building IVAPs and CVAPs, respectively. IVAPs split the training set $\mathcal{D}_{\text{train}}$ into a “proper” training set $\mathcal{D}'_{\text{train}}$ and a calibration set \mathcal{D}_{cal} . First, we fit the ML classifier to the proper training set. Then, we fit two Isotonic Regression algorithms (f_0 and f_1 in Algorithm 1) to the prediction scores in the calibration set and the predicted score on the test sample. IVAPs output two probability values, each one assuming that the ground-truth value for the test sample is either 0 or 1. The intuition is that, since the ground truth for the test sample is either 0 or 1, one of the two probabilities will be calibrated.

Algorithm 1 Inductive Venn-Abers Predictor (IVAP)

Require: Scoring ML classifier f (e.g., XGBoost), proper training dataset $\mathcal{D}'_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_{\text{train}}}$, calibration dataset $\mathcal{D}_{\text{cal}} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{n_{\text{cal}}}$, test example \mathbf{x}_{test}

- 1: $\hat{f} \leftarrow f.\text{fit}(\mathcal{D}_{\text{train}})$
- 2: $\{s_i\}_{i=1}^{n_{\text{cal}}} \leftarrow \{\hat{f}(\mathbf{x}_j)\}_{i=1}^{n_{\text{cal}}}$ // compute calibration scores
- 3: $s_{\text{test}} \leftarrow \hat{f}(\mathbf{x}_{\text{test}})$ // compute test score
- 4: $f_0 \leftarrow \text{Isotonic.fit}((s_0, y_0), \dots, (s_{n_{\text{cal}}}, y_{n_{\text{cal}}}), (s_{\text{test}}, 0))$
- 5: $f_1 \leftarrow \text{Isotonic.fit}((s_0, y_0), \dots, (s_{n_{\text{cal}}}, y_{n_{\text{cal}}}), (s_{\text{test}}, 1))$
- 6: $(p_0, p_1) \leftarrow (f_0(s_{\text{test}}), f_1(s_{\text{test}}))$
- 7: **return** (p_0, p_1)

Algorithm 2 Cross Venn-Abers Predictor (CVAP)

Require: Scoring ML classifier f , training dataset $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_{\text{train}}}$, test example \mathbf{x}_{test} , number of folds k

- 1: Split the training set $\mathcal{D}_{\text{train}}$ into k folds $\mathcal{D}_1, \dots, \mathcal{D}_k$
- 2: **for** $i \leftarrow 1$ to k **do**
- 3: $(p_0^i, p_1^i) \leftarrow \text{IVAP}(f, \mathcal{D}_{\text{train}} \setminus \mathcal{D}_i, \mathcal{D}_i, \mathbf{x}_{\text{test}})$
- 4: **end for**
- 5: **return** $\text{gmean}(\mathbf{p}_1) / (\text{gmean}(1 - \mathbf{p}_0) + \text{gmean}(\mathbf{p}_1))$

Note that p_0 and p_1 are not complementary, i.e., $p_1 \neq 1 - p_0$. In fact, it always holds that $p_0 < p_1$. In practice, for reasonably-sized datasets, p_0 and p_1 will have similar values [10]. Because of this, if one of the two is perfectly calibrated, we can expect that their ‘‘average’’ will be well-calibrated. As such, we merge the two probabilities p_0 and p_1 into a single value, as follows:

$$p = \frac{p_1}{1 - p_0 + p_1}. \quad (3)$$

Choosing this value of p yields log-minimax IVAPs, that is, it minimizes the regret of using p instead of the appropriate p_0 or p_1 over the log-loss. We point the reader to a complete proof of the calibration of IVAPs [34] and the log-minimax rule for deriving p [9].

CVAPs are an extension of IVAPs, dropping the theoretical guarantees on calibration in favor of a stronger empirical performance [9]. The algorithm splits the training set into k non-overlapping folds and applies IVAP k times, each time considering one fold as the calibration set and the remainder folds as the proper training set. This procedure results in two vectors $(\mathbf{p}_0, \mathbf{p}_1)$, where (p_0^i, p_1^i) are the outputs of IVAP considering the i -th fold as calibration set. We then merge the vectors into a single probability value, as follows:

$$p = \frac{\text{gmean}(\mathbf{p}_1)}{\text{gmean}(1 - \mathbf{p}_0) + \text{gmean}(\mathbf{p}_1)}, \quad (4)$$

where $\text{gmean}(\cdot)$ indicates the geometric mean. Note that, Eq. (3) is a special case of Eq. (4) when $k = 1$. As for Eq. (3), it can be demonstrated that choosing p as in Eq. (4) minimizes the regret over the log-loss, i.e., is log-minimax [9].

Multiclass VAPs. So far, we have discussed VAPs in the binary classification case. We now outline different techniques for generalizing these algorithms to the multiclass case.

A first approach consists of treating the multiclass problem into multiple one-versus-all (binary) classification problems, and constructing a VAP for each class [35]. However, in our

specific application scenario (selective classification), we are not interested in calibrating *every* predicted probability, but only the probability of the predicted class. To this end, we follow the guidelines of prior work [36] and apply VAPs only for calibrating the probability that the predicted class is correct. To do so, we relabel each instance in the calibration set to 1 if the ground-truth class is equal to the predicted class of the model, and 0 otherwise. Formally, with reference to Algorithms 1 and 2, after splitting the data in training and calibration set, we construct a new calibration set $\mathcal{D}_{\text{cal}}^{\text{bin}} = \{(\mathbf{x}_i, y_i^{\text{bin}})\}_{i=1}^{n_{\text{cal}}}$ with binary labels, as follows:

$$y_i^{\text{bin}} = \begin{cases} 1 & \text{if } y_i = \arg \max \hat{f}(\mathbf{x}_i), \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

The remainder of the IVAP and CVAP algorithms can be executed without further modifications. The final result is a VAP that calibrates the probability of correct classification, which can then be leveraged for ASAP classification.

Computational complexity of VAPs: we remark that building IVAPs and CVAPs as per Algorithms 1 and 2 appears computationally onerous, as it requires fitting two isotonic regression model for each test example. However, we note that the two isotonic regression models are fitted to the calibration scores plus only one other example, namely, the test example. One can leverage this property to reduce the computational complexity of the calibration procedure (lines 4-6 of Algorithm 1) to $O(n_{\text{cal}} \log(n_{\text{cal}}))$, where n_{cal} is the number of examples in the calibration set. The remainder of the total complexity is dominated by model training (line 1 of Algorithm 1), which is performed only once and in an offline phase. This makes the application of VAPs feasible for our application scenario, where the ML model must supply a prediction every second. Deriving the efficient algorithm is non-trivial, therefore, we omit the proof for brevity. We refer the readers to a detailed derivation [9] and a reference Python implementation of efficient IVAPs [37].

V. ILLUSTRATIVE NUMERICAL RESULTS

We now discuss illustrative numerical results highlighting the practical advantages brought by our ASAP classifier compared to the state-of-the-art. In particular, we empirically show that our ASAP classifier can:

- 1) Satisfy (on average over 10-fold cross-validation splits) a user-specified high probability of correct classification (95% or 99%), while rejecting a reasonable number of test samples (on average less than 10% for achieving $> 95\%$ selective accuracy)
- 2) Achieve $\sim 8x$ speedups on the mean time-to-predict compared to a state-of-the-art failure-cause classification aggregate strategy operating on 15-minute windows, while maintaining the same level of accuracy ($> 95\%$)

We consider a single XGBoost model with default configuration as an ML classifier [13]. This is because gradient-boosted tree models are the current state-of-the-art in ML for tabular data [38], [39]. We borrow from *VennABERS.py* [37] for implementing IVAPs and CVAPs. Results are aggregated

TABLE I
PROBABILISTIC PREDICTION METRICS FOR UNCALIBRATED, IVAP AND CVAP CLASSIFIERS. WE REPORT MEAN AND STANDARD DEVIATIONS OVER 10-FOLD CROSS-VALIDATION.

Metric	Uncalibrated	IVAP	CVAP
Log-loss	0.21 ± 0.06	0.14 ± 0.04	0.13 ± 0.03
Brier score	0.09 ± 0.02	0.04 ± 0.01	0.039 ± 0.009
ECE	0.028 ± 0.015	0.025 ± 0.012	0.023 ± 0.009

over 10-fold cross-validation. To avoid data leakage, we perform train-test splits over disjoint 15-minute windows.

As a baseline algorithm, we consider an Uncalibrated ASAP classifier trained on the expanding-window dataset that thresholds the XGBoost class scores without applying any post-processing (Section IV-B). Note that, while CVAP and IVAP require holding out a portion of the training set for calibration, we train the uncalibrated classifier on the whole training set for a fair comparison. For IVAP, we hold out 20% of the training dataset for calibration. For CVAP, we consider splitting the training dataset into $k = 5$ folds.

We first quantitatively evaluate whether or not IVAPs and CVAPs provide better probabilistic predictions compared to the Uncalibrated model. We report three metrics for probabilistic prediction: i) log-loss, ii) Brier score, and iii) Expected Calibration Error (ECE). The log-loss is defined as follows:

$$\text{Log-loss} = \begin{cases} -\log(p) & \text{if correct,} \\ -\log(1-p) & \text{otherwise,} \end{cases} \quad (6)$$

where p is the probability associated to the predicted class. The Brier score is defined as follows:

$$\text{Brier score} = \begin{cases} (p-1)^2 & \text{if correct,} \\ ((1-p)-1)^2 & \text{otherwise.} \end{cases} \quad (7)$$

The Brier score can be interpreted as the mean squared error applied to predicted probabilities. We remark that both the log-loss and the Brier score are *strictly proper scoring rules* [40], that is, lower scores indicate a better approximation of the true data distribution. Finally, ECE is defined as the weighted average of the absolute difference between the mean of the predicted probabilities (mop) and the fraction of correct predictions (foc) in $M = 10$ equally-spaced bins (the same computation that resulted in Fig. 5), as follows:

$$\text{ECE} = \sum_{i=1}^M \frac{|B_i|}{n_{\text{test}}} |\text{foc}(B_i) - \text{mop}(B_i)|. \quad (8)$$

We underline that, unlike the log-loss and the Brier score, the ECE is not a proper scoring rule, that is, lower ECE *does not* generally imply better probabilistic predictions. As a simple counterexample, a classifier always predicting the class frequencies in the training set will achieve near-zero ECE, despite being useless for making predictions. Still, we report the metric for its useful intuitive interpretation.

Table I illustrates the average log-loss, Brier score, and ECE of the Uncalibrated XGBoost model, IVAP, and CVAP. We observe that VAPs achieve better values for all the considered metrics. In particular, since VAPs achieve better log-loss and Brier score, we can conclude that they can provide better probabilistic predictions than the Uncalibrated model.

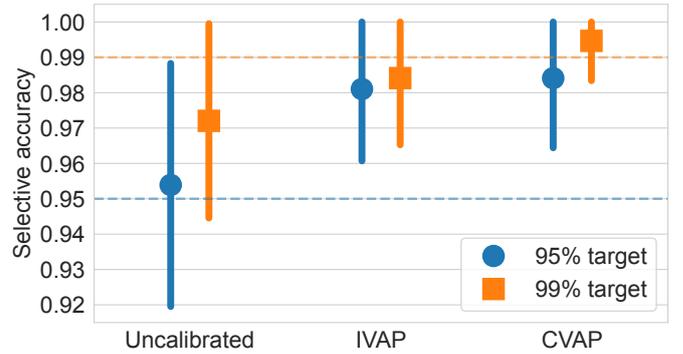


Fig. 6. Selective accuracy of Uncalibrated, IVAP and CVAP for thresholds 95%, 99% on the probability of correct classification. We report means and standard deviations over 10-fold cross-validation.

TABLE II
SELECTIVE ACCURACY AND REJECTION RATIOS OF UNCALIBRATED, IVAP AND CVAP FOR ASAP CLASSIFICATION. WE REPORT MEAN AND STANDARD DEVIATIONS OVER 10-FOLD CROSS-VALIDATION.

Metric	Uncalibrated	IVAP	CVAP
Target: > 0.95 selective accuracy			
Selective accuracy	0.954 ± 0.033	0.981 ± 0.019	0.984 ± 0.019
Rejection ratio	0.3% ± 0.5%	5% ± 3%	8% ± 4%
Target: > 0.99 selective accuracy			
Selective accuracy	0.972 ± 0.026	0.984 ± 0.018	0.994 ± 0.011
Rejection ratio	1.0% ± 0.9%	11% ± 6%	18% ± 6%

We now evaluate our approach in terms of *selective accuracy*, that is, the frequency of correct classification of the accepted predictions. Fig. 6 illustrates the selective accuracy of the ASAP model under error rate constraints of $\alpha = 0.05$ and $\alpha = 0.01$ (i.e., 95% and 99% accuracy, respectively). For the 95% case, we observe that even though the uncalibrated model delivers, on average, a 95% selective accuracy, it does not perform consistently among different splits. Indeed, the minimum selective accuracy is below 91%, illustrating that the uncalibrated model tends to be overconfident. From a failure management perspective, this can result in underestimating the risk associated with accepting wrong failure classifications, which may have a devastating impact if said classifications are leveraged for choosing a mitigation strategy. Conversely, both IVAP and CVAP deliver a selective accuracy always greater than 0.95%. For the 99% case, we observe that both the uncalibrated model and IVAP do not meet the target and deliver an accuracy less than 99%. In contrast, CVAP delivers an average selective accuracy above 99% with relatively low variance across the splits. This is a remarkable result, as our training dataset is relatively small, and we are querying the extreme tail of the predictive distribution. We draw two main conclusions from this first analysis. First, calibrating via either IVAP or CVAP consistently improves the selective accuracy over the uncalibrated model. Second, consistent with prior findings [9], we conclude that although CVAPs drop the theoretical guarantees of IVAPs, they provide better empirical performance. From the point of view of network management, the model's predictions provided by our framework are highly reliable for supporting decision-making.

We complement the comparisons in Fig. 6 with Table II, which reports the percentage of rejected test samples

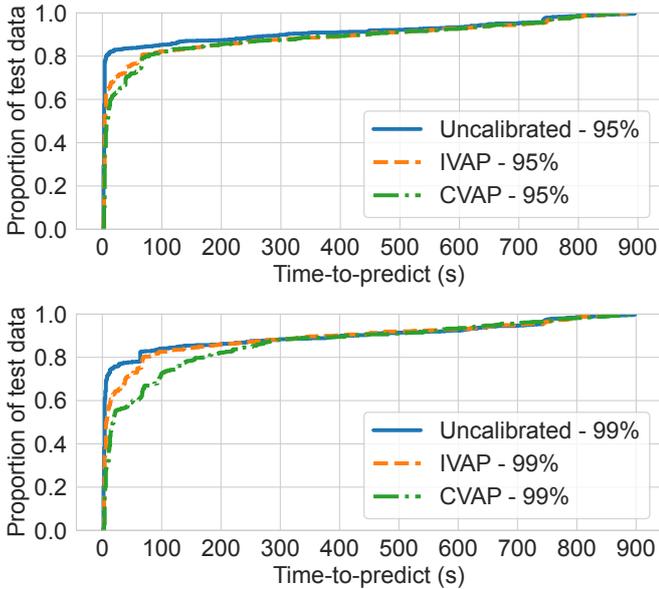


Fig. 7. Time-to-predict CDFs for Uncalibrated, IVAP and CVAP for selective classification under 95%, 99% probability of correct prediction.

for Uncalibrated, IVAP and CVAP for thresholds 95%, and 99% on the probability of correct classification. As expected, the more stringent the threshold is, the greater the rejection ratio. We observe that the uncalibrated model yields very low rejection rates, a few percent on average; however, as anticipated before, this comes at the price of a selective accuracy that is, on average, lower than the safety threshold. In contrast, IVAPs and CVAPs achieve a reasonable rejection ratio (less than 10% and 20% of the total test samples for 95% and 99% selective accuracy, respectively) while providing well-calibrated predictive probabilities. Finally, we comment on why IVAPs and CVAPs yield rejection rates relatively high compared to the uncalibrated model. Recall that the underlying ML models in IVAP and CVAP are trained with 20% less training data than the uncalibrated model, as they require a held-out calibration set. In other words, they have less predictive power than the uncalibrated classifier. As such, decisions must be more conservative to satisfy the selective accuracy constraint, i.e., the model will reject more test samples. Sacrificing predictive power in favor of calibration is a trade-off that needs to be carefully evaluated when leveraging VAPs. We argue that in a high-risk application such as failure management, well-calibrated probabilistic predictions are more useful for decision-making than hard predictions with no uncertainty quantification. We expect rejection rates to diminish in a scenario of relative data abundance (e.g., thousands of observations), where subtracting 20% of the training set for calibration has a negligible impact on the underlying ML model’s performance.

We now quantitatively assess the improvements brought by our ASAP framework in terms of time-to-predict. Fig. 7 illustrates the cumulative distribution of the time elapsed before the model outputs an $(1 - \alpha)$ -confident prediction for $\alpha = 0.05$ and $\alpha = 0.01$. We see that, in general, most predictions are provided already at the first second. This is expected, since a baseline model operating only on one-second telemetry already displayed accuracy close to 90%. Moreover,

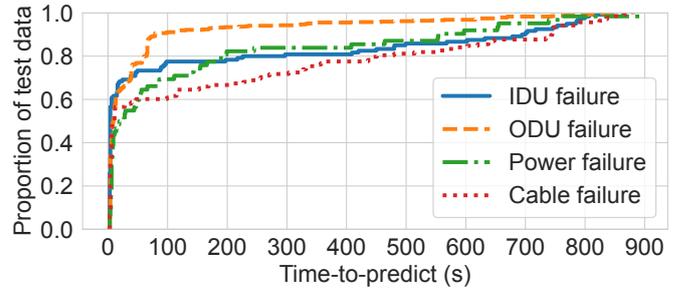


Fig. 8. Per-class time-to-predict CDFs of the CVAP classifier under a 95% constraint on the probability of correct prediction.

we observe that the uncalibrated model outputs predictions on average earlier than IVAP and CVAP. This, however, comes at the price of an inferior selective accuracy, as previously discussed for Fig. 6. Conversely, IVAPs and CVAPs can output the majority of predictions in a few minutes, while respecting the safety threshold. In particular, we observe that, for the 99% case, the average time-to-predict increases compared to 95%. This confirms that the model can indeed delay its prediction until it matures a sufficient level of confidence to satisfy the safety threshold. In particular, the CVAP classifier yields an average time-to-predict equal to 101s and 119s for 95% and 99% accuracy targets, respectively. This grants a mean speedup of 8.9x and 7.5x, respectively, compared to the state-of-the-art aggregate classifier operating on 15-minute windows. We conclude that our ASAP classifier can provide both reliable and fast predictions.

Fig. 8 breaks down Fig. 7 on each individual failure class for the CVAP classifier under a 95% selective accuracy constraint. Note that, in this case, the y-axis reports the proportion of test data per class. We observe that some failure classes are predicted on average later than others. In particular, we observe that, cable failures are predicted on average later than all other failures. This is an expected result, considering the class distribution in our dataset. Recall that the cable failure is the least represented among all the failure classes with only 75 observation (6.6x less than ODU failure, the most represented class). It is a well-known result that, in scenarios with class imbalance, a ML model will tend to favor the better-represented classes [41], [42]. For this reason, we can expect the predictions for the cable class to be, on average, more uncertain compared to the other failure classes, resulting in a longer time-to-predict on average. The information provided by the above analysis can also be leveraged in a counterfactual manner. For example, in case the ML model outputs several consecutive predictions with high uncertainty, regardless the output classes in these predictions, one can conclude that the most likely class is among those that are typically predicted later, e.g., cable failures, according to our analysis.

VI. CONCLUSION

In this paper, we introduced As-Soon-As-Possible (ASAP) selective classification for hardware-failure-cause identification in microwave networks. In contrast to the current state-of-the-art, which leverages data collected in fixed-size measurement windows, our ASAP classification framework is designed to output a prediction as soon as the probability

of correct classification exceeds a user-specified threshold. To this end, we leverage recent advances in the field of Venn-Abers predictors, which allows to turn any scoring classifier into a well-calibrated probabilistic classifier. Overall, our framework empowers the network manager with prompt and reliable failure-cause predictions, reducing the time-to-predict by $\sim 8x$ while ensuring a selective accuracy greater than 95%. Future research directions include the investigation of feature importance in determining the selective accuracy of the prediction, i.e., to identify whether using or removing any alarm from the features set has a positive or a negative impact on the time-to-predict.

REFERENCES

- [1] C.-X. Wang *et al.*, “On the road to 6g: Visions, requirements, key technologies, and testbeds,” *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 905–974, 2023.
- [2] M. Banafaa *et al.*, “6g mobile communication technology: Requirements, targets, applications, challenges, advantages, and opportunities,” *Alexandria Engineering Journal*, vol. 64, pp. 245–274, 2023.
- [3] Z. Qadir, K. N. Le, N. Saeed, and H. S. Munawar, “Towards 6G internet of things: Recent advances, use cases, and open challenges,” *ICT Express*, vol. 9, no. 3, pp. 296–312, 2023.
- [4] J. Mata *et al.*, “Artificial intelligence (ai) methods in optical networks: A comprehensive survey,” *Optical switching and networking*, vol. 28, pp. 43–57, 2018.
- [5] M. F. Silva, A. Pacini, A. Sgambelluri, and L. Valcarenghi, “Learning long-and short-term temporal patterns for ml-driven fault management in optical communication networks,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2195–2206, 2022.
- [6] L. Gupta *et al.*, “Fault and performance management in multi-cloud virtual network services using ai: A tutorial and a case study,” *Computer Networks*, vol. 165, p. 106950, 2019.
- [7] J. Gallego-Madrid *et al.*, “Machine learning-based zero-touch network and service management: A survey,” *Digital Communications and Networks*, vol. 8, no. 2, pp. 105–123, 2022.
- [8] D. Wang *et al.*, “A review of machine learning-based failure management in optical networks,” *Science China Information Sciences*, vol. 65, no. 11, p. 211302, 2022.
- [9] V. Vovk, I. Petej, and V. Fedorova, “Large-scale probabilistic predictors with and without guarantees of validity,” in *Advances in Neural Information Processing Systems*, vol. 28, 2015.
- [10] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic Learning in a Random World*. Berlin, Heidelberg: Springer-Verlag, 2005.
- [11] Z. Ruan *et al.*, “Microwave link failures prediction via lstm-based feature fusion network,” in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.
- [12] O. Ayoub *et al.*, “Explainable artificial intelligence in communication networks: A use case for failure identification in microwave networks,” *Computer Networks*, vol. 219, p. 109466, 2022.
- [13] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, p. 785–794.
- [14] F. Lateano, O. Ayoub, F. Musumeci, and M. Tornatore, “Machine-learning-assisted failure prediction in microwave networks based on equipment alarms,” in *2023 19th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE, 2023, pp. 1–7.
- [15] M. Nouioua *et al.*, “A survey of machine learning for network fault management,” *Machine Learning and Data Mining for Emerging Trend in Cyber Dynamics: Theories and Applications*, pp. 1–27, 2021.
- [16] F. Musumeci, C. Rottondi, G. Corani, S. Shahkarami, F. Cugini, and M. Tornatore, “A tutorial on machine learning for failure management in optical networks,” *Journal of Lightwave Technology*, vol. 37, no. 16, pp. 4125–4139, 2019.
- [17] R. Gu, Z. Yang, and Y. Ji, “Machine learning for intelligent optical networks: A comprehensive survey,” *Journal of Network and Computer Applications*, vol. 157, p. 102576, 2020.
- [18] M. Bubniak, P. Musil, and P. Miłynek, “Application for an early detection of transmission parameters degradation of point-to-point microwave links,” in *2020 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE, 2020, pp. 82–86.
- [19] D. Jacoby, J. Ostrometzky, and H. Messer, “Short-term prediction of the attenuation in a commercial microwave link using lstm-based rnn,” in *2020 28th European Signal Processing Conference (EUSIPCO)*. IEEE, 2021, pp. 1628–1632.
- [20] —, “Model-based vs. data-driven approaches for predicting rain-induced attenuation in commercial microwave links: A comparative empirical study,” in *2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023, pp. 1–5.
- [21] F. Musumeci *et al.*, “Supervised and semi-supervised learning for failure identification in microwave networks,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1934–1945, 2020.
- [22] M. Choi, T. Kim, J. pil Lee, and S. Koh, “An empirical study on root cause analysis and prediction of network failure using deep learning,” in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2021, pp. 741–746.
- [23] L. Pan *et al.*, “Proactive microwave link anomaly detection in cellular data networks,” *Computer Networks*, vol. 167, p. 106969, 2020.
- [24] T. Tandel, O. Ayoub, F. Musumeci, C. Passera, and M. Tornatore, “Federated-learning-assisted failure-cause identification in microwave networks,” in *2022 12th International Workshop on Resilient Networks Design and Modeling (RNDM)*. IEEE, 2022, pp. 1–7.
- [25] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, “pforest: In-network inference with random forests,” 2022.
- [26] Understanding ITU-T error performance recommendations. https://www.julesbartow.com/Pictures/ITS/ITU-T_Errors_ApplicationNote2.pdf.
- [27] SIAE microwave product portfolio. <https://www.siaemic.com/index.php/products-services/telecommunication-systems/microwave-product-portfolio>.
- [28] A. Niculescu-Mizil and R. Caruana, “Predicting good probabilities with supervised learning,” in *Proceedings of the 22nd International Conference on Machine Learning*, 2005, p. 625–632.
- [29] M. Minderer *et al.*, “Revisiting the calibration of modern neural networks,” in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021.
- [30] J. Platt, “Probabilistic outputs for support vector machines and comparison to regularized likelihood methods,” in *Advances in Large Margin Classifiers*, vol. 10, no. 3, 1999, pp. 61 – 74.
- [31] M. Ayer *et al.*, “An Empirical Distribution Function for Sampling with Incomplete Information,” *The Annals of Mathematical Statistics*, vol. 26, no. 4, pp. 641 – 647, 1955.
- [32] A. Gammerman, V. Vovk, and V. Vapnik, “Learning by transduction,” in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI’98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, p. 148–155.
- [33] J. Lei and L. Wasserman, “Distribution-free Prediction Bands for Non-parametric Regression,” *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 76, no. 1, pp. 71–96, 07 2013.
- [34] I. Nourtdinov *et al.*, “Inductive Venn-Abers predictive distribution,” in *Proceedings of the Seventh Workshop on Conformal and Probabilistic Prediction and Applications*, vol. 91, 11–13 Jun 2018, pp. 15–36.
- [35] V. Manokhin, “Multi-class probabilistic classification using inductive and cross Venn-Abers predictors,” in *Proceedings of the Sixth Workshop on Conformal and Probabilistic Prediction and Applications*, vol. 60, 2017, pp. 228–240.
- [36] U. Johansson *et al.*, “Calibrating multi-class models,” in *Proceedings of the Tenth Symposium on Conformal and Probabilistic Prediction and Applications*, vol. 152, 08–10 Sep 2021, pp. 111–130.
- [37] P. Toccaceli, “VennABERS.py,” <https://github.com/ptocca/VennABERS>, 2022, [Online; accessed 7-Dec-2022].
- [38] R. Schwartz-Ziv *et al.*, “Tabular data: Deep learning is not all you need,” in *8th ICML Workshop on Automated Machine Learning*, 2021.
- [39] L. Grinsztajn, E. Oyallon, and G. Varoquaux, “Why do tree-based models still outperform deep learning on typical tabular data?” in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.
- [40] T. Gneiting and A. E. Raftery, “Strictly proper scoring rules, prediction, and estimation,” *Journal of the American statistical Association*, vol. 102, no. 477, pp. 359–378, 2007.
- [41] G. Lemaître *et al.*, “Imbalanced-learn: a python toolbox to tackle the curse of imbalanced datasets in machine learning,” *Journal of Machine Learning Research*, vol. 18, no. 1, p. 559–563, jan 2017.
- [42] M. A. M. Mateusz Buda, Atsuto Maki, “A systematic study of the class imbalance problem in convolutional neural networks,” *Neural Networks*, vol. 106, pp. 249–259, 2018.

5 | Machine Learning for Service Orchestration

In this Chapter, we propose using ML, in particular RL, for learning to dynamically assign incoming service requests to the appropriate computational resources. Differently from Chapter 3, we now consider an online optimization problem where service requests arrive and depart stochastically. In this context, in Section 5.1 we consider training a RL-based orchestrator that can scale to networks different, and possibly much larger, than the ones seen during training. We propose solving this problem by using set-based neural networks, and our numerical results illustrate the excellent generalization capabilities of our solution. Following up, in Section 5.2 we formulate online service orchestration as a multi-objective optimization problem, and we propose using Multi-Objective RL to derive a Pareto Front of policies achieving locally-optimal trade-offs. Numerical results based on the profiling of real-world edge devices show that our Multi-Objective RL solution discovers better trade-offs than conventional heuristics.

5.1. Scalable Service Orchestration with Reinforcement Learning

In future communication networks, the integration of edge and cloud resources is expected to form a computing continuum where applications are executed on different hardware infrastructure according to their service requirements [41]. In this context, here we consider the problem of dynamically assigning services to compute nodes to jointly optimize two competing objectives, namely, blocking probability and service delay. Our main contribution lies in designing a neural network architecture that can generalize and scale to a number of edge nodes much larger than training, and possibly evolving with time. We achieve this goal by formulating service orchestration as a problem over *sets* of edge nodes, and by leveraging Deep Sets [43] neural networks, an improved model over the one we used in the DeepLS paper [35]. In contrast to conventional neural networks, i.e., MLPs, which learn functions over fixed-dimensional vector spaces, Deep Sets learn set functions,

and can be applied to input sets of arbitrary size (including sets much larger than the ones seen during training). Illustrative experimental results illustrate the advantage of our design, with our Deep Sets RL policies consistently outperforming conventional heuristics and generalizing their learned orchestration logic on networks several times larger than training. Moreover, as Deep Sets can be massively parallelized, our solution attains inference times less than 1ms on commodity hardware, illustrating its potential for near real-time decision-making.

DRL-FORCH: A Scalable Deep Reinforcement Learning-based Fog Computing Orchestrator

Nicola Di Cicco*, Gaetano Francesco Pittalà†, Gianluca Davoli†, Davide Borsatti†,
Walter Cerroni†, Carla Raffaelli†, Massimo Tornatore*

*Department of Electronics, Information, and Bioengineering (DEIB), Politecnico di Milano, Italy

†Department of Electrical, Electronic, and Information Engineering, University of Bologna, Italy

Abstract—We consider the problem of designing and training a neural network-based orchestrator for fog computing service deployment. Our goal is to train an orchestrator able to optimize diversified and competing QoS requirements, such as blocking probability and service delay, while potentially supporting thousands of fog nodes. To cope with said challenges, we implement our neural orchestrator as a Deep Set (DS) network operating on sets of fog nodes, and we leverage Deep Reinforcement Learning (DRL) with invalid action masking to find an optimal trade-off between competing objectives. Illustrative numerical results show that our Deep Set-based policy generalizes well to problem sizes (i.e., in terms of numbers of fog nodes) up to two orders of magnitude larger than the ones seen during the training phase, outperforming both greedy heuristics and traditional Multi-Layer Perceptron (MLP)-based DRL. In addition, inference times of our DS-based policy are up to an order of magnitude faster than an MLP, allowing for excellent scalability and near real-time online decision-making.

Index Terms—Fog Computing, Reinforcement Learning, Orchestration, Optimization, Deep Learning

I. INTRODUCTION

Fog Computing (FC) is a distributed computing paradigm that extends Cloud Computing (CC) by offloading tasks to devices situated in close proximity to the user, making use of their computing, storage, and communication resources [1]. FC can provide, among other benefits, lower latency for service access and fruition compared to CC. As FC is meant to handle heterogeneous devices with context-specific constraints (e.g., limited power, unstable connectivity, etc.), an orchestration layer, represented by an entity referred to as the fog orchestrator, is required to oversee the management of resources and the deployment of services. As such, a proper node selection policy is crucial in the effectiveness of FC systems, as multiple factors must be considered, including but not limited to the available computing power and the proximity of the service to the consumer.

Deep Reinforcement Learning (DRL) is a rapidly growing field that has the potential to revolutionize the way FC networks are optimized and controlled. Thanks to its capability of learning without explicit human supervision, DRL has found broad applications in networking, such as in traffic control [2], routing optimization [3], [4], and resource allocation [5], [6].

However, one severe limitation of traditional DRL-based approaches, often based on Multi-Layer Perceptron (MLP)

neural networks, is their inability to deal with variable-sized input and output spaces. For instance, in the context of service orchestration in FC, the same DRL-based orchestrator must be applicable to clusters of fog nodes of arbitrary size, as the number of fog nodes is bound to change over time due to random connections/disconnections. Unfortunately, since the input and output dimensionalities of MLPs are defined on fixed-dimensional vector spaces, it is often impossible to generalize DRL-based approaches without retraining from scratch. These limitations may prevent the deployment of DRL-approaches in real application scenarios.

In this work, to cope with this fundamental challenge, we formulate the problem of online service orchestration in FC networks as a Machine Learning problem defined on sets. Indeed, we argue that a cluster of fog nodes is better represented as a set, i.e., as an arbitrary-size collection of distinct elements, rather than as a fixed-dimensional vector representation (as in traditional MLP-based approaches). Following this intuition, we leverage Deep Sets (DS) neural networks [7] to implement our DRL-based policy. DS are a family of neural networks tailored for processing data structured as sets. Once trained, DS networks can perform inference to sets of arbitrary size without the need for retraining, with a computational complexity linear in the number of elements in the input set. Therefore, integrating DS into FC networks allows for learning generalizable policies independently of the number of nodes available at a particular time in the network.

As a reference application scenario, we build upon the state-of-the-art FORCH orchestrator [8], a service orchestration system specifically designed for flexibility, which extends the Everything-as-a-Service (XaaS) model to fog computing environments. At each service request arrival, telemetry data together with service specifications are fed to a DRL-based policy, which selects the best node for serving the request. The choice of the nodes is driven by a properly-shaped reward signal, which in this work aims to find an optimal trade-off between blocking probability and requested service latency. Our illustrative numerical results on a simulation environment show that our DS-based policy is able to generalize its knowledge for numbers of fog nodes up to two orders of magnitude larger than training, outperforming both standard greedy heuristics and MLP-based policies. We, therefore, lay the foundations for implementing DRL-based intelligence in real orchestration software.

N. Di Cicco and G. F. Pittalà are co-first authors. Our source code is publicly available at <https://github.com/bonsai-lab-polimi/netsoft2023-drl-forch>.

II. RELATED WORK

DRL, thanks to its capability of optimizing long-term objectives in dynamical systems, has found broad application in the context of resource allocation in FC networks. In this Section, we briefly survey recent literature on the topic and highlight our main contributions compared to the state-of-the-art.

Literature sharing the most similarity with our work considers the problem of resource allocation in FC networks. In [9], authors investigate the task of placing service containers in fog nodes given a list of service demands to minimize the number of deployed containers and optimize QoS metrics. An MLP-based Deep-Q Learning (DQN) agent is proposed, outperforming traditional optimization methods. In [10], authors consider the task of allocating service requests in a Fog Radio Access Network (F-RAN). An MLP-based DQN agent is proposed to optimize the utilization of edge resources, the utility of the served requests, and the acceptance probability. In [11], authors propose an intelligent Reinforcement Learning (RL)-based resources management at the network controller side, for sustainable Fog Radio Access Networks. Authors devise an MLP-based DQN to minimize the energy consumption in the network. In [12], authors propose three MLP-based DRL scheduling algorithms for the cloud-fog continuum, minimizing both the makespan and processing cost of workflows.

A second relevant body of literature considers the task offloading/migration/caching problem in FC networks. In [13], authors deal with the issues of content caching strategy, computation offloading policy, and radio resource allocation in Fog Computing networks, with the objective of minimizing the end-to-end delay. Authors develop an MLP-based DRL algorithm for solving the proposed optimization problem. In [14], authors focus their attention on offloading device-to-device issues in FC industrial applications. In particular, they make use of two different scheduling algorithms based on RL called Dynamic Reinforcement Learning Scheduling (DRLS) and Deep Dynamic Scheduling (DDS), showing how these two algorithms can drastically reduce energy costs compared to other offloading/non-offloading schemes. In [15], authors consider the problem of many-to-many task offloading in a dynamic vehicular environment. In particular, they adopt a Multi-Agent Gated actor Attention Critic (MA-GAC) approach, leading to an efficient offloading optimization process in a distributed manner. In [16], authors propose a decentralized task offloading method based on Transformer and Policy Decoupling-based Multi-Agent Actor-Critic (TPDMAAC), highlighting the flexibility of this algorithm, as it can be adapted to other scenarios by the fine-tuning of its parameters even with an uncertain load in the edge server. In [17], authors present a new component migration strategy in an NFV-based hybrid cloud/fog system considering the mobility of both end users and fog nodes. In particular, they propose a DRL approach, based on a Double Deep-Q Network (DDQN), to decide where and when to migrate application components, achieving better results in both delay and power consumption compared to other state-of-the-art application migration strategies. In [18],

authors deal with the cooperative edge caching problem in F-RANs to minimize the content transmission delay. Authors propose a Multi Agent Reinforcement Learning (MARL)-based cooperative caching scheme, that applies a DDQN on each Fog Access Point (F-AP).

While all of the aforementioned works comprise significant advances in the application of DRL in FC environments, they do not address the problem of zero-shot generalization on unseen (and possibly more complex) deployment environments. Most state-of-the-art DRL approaches for FC make use of MLP deep neural networks for implementing their DRL policy. As MLPs operate on fixed-dimensional vector spaces, it is impossible to generalize their learned knowledge to different deployment scenarios without retraining. Multi-agent approaches such as in [15], [18] partially address the scalability issue of state/action spaces by distributing control over multiple learning agents. However, a distributed MARL-based approach is fundamentally incompatible with centralized service orchestration, and limitations in zero-shot generalization due to fixed input/output dimensionalities of MLPs still stand. In contrast, [16] explicitly considers the problem of fixed input/output dimensionality of MLPs, but their countermeasure still requires fine-tuning when applied to testing scenarios different than training. Overall, the intrinsic difficulty of zero-shot generalization makes deployment of DRL-based orchestrators in real scenarios problematic.

In this work, to cope with this scalability challenge, we consider online service orchestration in FC networks as a Machine Learning problem defined on sets of fog nodes. We make use of deep neural network architectures tailored for processing data structured as sets, enabling generalization to numbers of fog nodes of arbitrary size, possibly significantly larger than training. To illustrate our findings, we build on the FORCH [8] orchestrator, abstract its core functionalities into online decision-making, and train via DRL a DS neural network to optimize the process of service allocation.

III. BACKGROUND

In this Section, we provide some background on the basics of DRL and on permutation-equivariant/invariant Deep Sets [7] neural networks, and we discuss how DS can be leveraged for learning DRL-based policies that can deal with variable state/action spaces without the need for retraining.

A. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a Machine Learning paradigm that targets sequential decision-making problems [19]. A DRL agent learns by trial-and-error via repeated interactions with a dynamic environment, with the objective of maximizing the accumulation of rewards. Specifically, a DRL environment is often modeled as a Markov Decision Process (MDP). An MDP consists in a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle$, where \mathcal{S} is the state space, \mathcal{A} is the action space, P is the transition probability matrix, R is the reward function, γ is the discount factor, and μ is the initial state distribution. The goal of DRL is to learn a neural network-based policy $\pi_\theta(a|s)$, $a \in \mathcal{A}$, $s \in \mathcal{S}$,

parameterized by θ , such that the discounted accumulation of rewards over a time-horizon T is maximized:

$$\theta = \arg \max_{\theta} J(\pi_{\theta}) = \mathbb{E} \left[\sum_{t=0}^T \gamma^t R_t \mid \pi_{\theta} \right] \quad (1)$$

Additionally, we define the discounted return at time step t as $G_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k$. Moreover, given a policy π , we define its value function $V_{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$.

While small-size MDPs can be solved to optimality with tabular RL methods, the state/action spaces of real-world MDPs may grow prohibitively large. In the context of FC, the resource occupation of fog nodes, which in its simplest form can be represented by a real number in $[0, 1]$, may take an infinite amount of possible values. As such, one needs to leverage function approximation via deep neural networks for solving large-size MDPs.

In the following, we will refer to Actor-Critic DRL algorithms, specifically to Proximal Policy Optimization (PPO) [20]. PPO is among the most widely used DRL algorithms thanks to its training efficiency and robustness to the choice of hyperparameters. Briefly, training Actor-Critic algorithms requires defining both a policy $\pi_{\theta}(a|s)$ and a value function approximator $V_{\xi}(s)$, parameterized by θ and ξ , respectively. Once the policy is trained, the value function approximator is discarded, and only the policy is kept for inference.

While traditional DRL-based approaches in FC employ MLP neural networks for implementing $\pi_{\theta}(a|s)$ and $V_{\xi}(s)$, the input/output spaces of said functions are limited by the dimensionality of state/action spaces in the training environment. For instance, if the dimensionality of the state/action space depends on the number of fog nodes, MLP networks cannot be applied without retraining on a different number of fog nodes, as in MLPs the dimensionality of the input space is hard-coded in the neural network architecture. In the following, we will tackle this problem by formulating the problem of resource allocation in FC as a problem defined on sets, and we will employ neural network architectures tailored for processing data structured as sets.

B. Deep Set Neural Networks

We now consider the case in which the input data to our neural network can be represented as a set. Formally, we consider our input as a set of n elements $X = \{x_1, \dots, x_n\}$. Our goal is to design neural networks that are insensitive to the ordering of elements in the set. Formally, we would like neural networks that are either permutation-equivariant or permutation-invariant with respect to the ordering of the elements in the input sets. We define a function f to be permutation-invariant if $f(X) = f(p(X))$ for any permutation p , i.e., if the output of the function does not depend on the ordering of the input set elements. Similarly, we define a function f to be permutation-equivariant if $f(p(X)) = p(f(X))$ for any permutation p , i.e., if the permutation on the input set elements is reflected on the output.

In this work, we choose Deep Sets (DS) [7] as the blueprint for implementing permutation-equivariant and permutation-

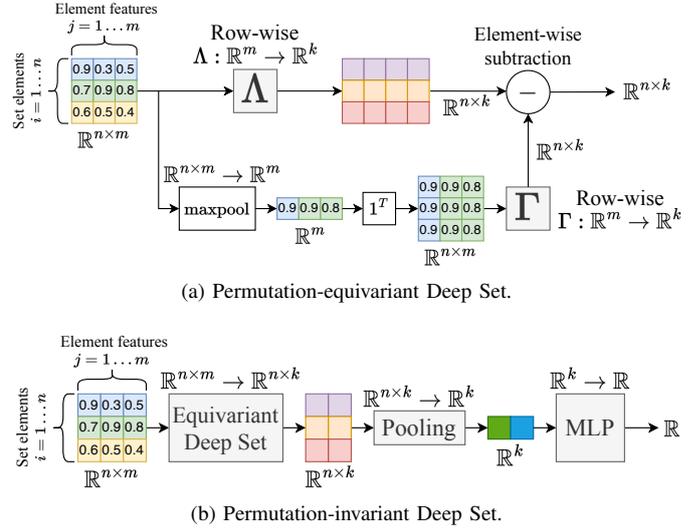


Fig. 1. Computational flow of permutation-equivariant and permutation-invariant Deep Set neural networks.

invariant deep neural networks. The properties of Deep Sets that make them an attractive choice for implementing scalable and intelligent fog orchestrators are: i) Deep Sets implement either permutation-equivariant or permutation-invariant neural networks, i.e., the learned functions are not bound to a specific indexing of the fog nodes; ii) the time complexity of Deep Sets scales linearly with the number of fog nodes, and computations can be trivially parallelized over different elements in the set; and iii) Deep Sets can infer on sets with an arbitrary number of elements, allowing generalization to different numbers of fog nodes. Formally, considering sets of n elements each one characterized by m features, Deep Sets realize a permutation-equivariant function $f(\mathbf{x}): \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^{n \times k}$ as follows:

$$f(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{\Lambda} - \mathbf{1}^T \mathbf{\Gamma} \text{maxpool}(\mathbf{x})) \quad (2)$$

where $\mathbf{x} \in \mathbb{R}^{n \times m}$ are the input features, $\mathbf{\Lambda}, \mathbf{\Gamma} \in \mathbb{R}^{m \times k}$ are trainable parameters, and $\sigma(\cdot)$ is an element-wise nonlinearity (e.g., ReLU). The above equation realizes a permutation-equivariant function with respect to the rows of \mathbf{x} . The product $\mathbf{x}\mathbf{\Lambda}$ consists in applying the same linear transformation $\mathbf{\Lambda}$ to each row of \mathbf{x} . The function maxpool extracts the maximum of each column of \mathbf{x} , thus being permutation-invariant. One can therefore stack multiple layers in the form of Eq. (2) to build Deep Set permutation-equivariant neural networks. The update rule of Deep Sets is closely related to Message Passing Graph Neural Networks (MPNNs) [21]. Briefly, Deep Sets can be seen as MPNNs where at each message-passing iteration all nodes receive the same message. Since in this work we do not assume an adjacency model among fog nodes, we found Deep Sets more suitable to our needs than the more general MPNNs. The computational flow of permutation-equivariant Deep Sets is illustrated in Fig. 1a.

The computations in Eq. (2) do not assume a fixed value for n , i.e., the matrix multiplications appearing in Eq. (2) can be performed irrespectively from the size of the input set.

As such, once a Deep Sets networks are trained, they can perform inference on input sets of arbitrary size. Moreover, their computational complexity is linear in the number of elements in the set, and computations can be parallelized over the elements in the set (i.e., the rows of \mathbf{x}) in a similar fashion to “batching” in deep neural networks.

In the context of Actor-Critic DRL algorithms, permutation-equivariant Deep Sets can be leveraged to implement a policy $\pi_\theta(a|s): \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^n$ for a discrete action space over the set elements. In particular, the state $s \in \mathbb{R}^{n \times m}$ represents a set of n elements each characterized by m features, and the action $a \in \mathbb{R}^n$ represents a categorical distribution over the n elements in the input set.

In a similar fashion, one can leverage permutation-invariant Deep Sets to implement the value function approximator $V_\xi(s): \mathbb{R}^{n \times m} \rightarrow \mathbb{R}$. In this case, one requires permutation invariance as the output of $V_\xi(s)$ is a scalar. To realize permutation-invariant Deep Sets, one can apply a permutation-invariant pooling operator $pool: \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^m$ to the final layer of an equivariant Deep Set network (e.g., max, sum or mean pooling). The pooling operator yields a fixed-size representation of an arbitrary-size set. Said representation can then be further transformed by a function $f: \mathbb{R}^m \rightarrow \mathbb{R}$ (e.g., a small MLP) to yield the desired output scalar. The computational flow of permutation-invariant Deep Sets is illustrated in Fig. 1b.

IV. DRL-FORCH: DEEP REINFORCEMENT LEARNING-BASED FOG ORCHESTRATOR

A. System Model

Our reference environment is the FORCH orchestration system [8], designed to provide services to users in a dynamic way, with a focus on resource utilization efficiency and service fruition latency. In principle, the orchestrator listens for service requests from the users and attempts to activate said services on the available fog nodes. In this context, one of the key issues is how to properly select the node on which to deploy the service. This problem is made even more delicate by the nature of the fog environment, where available nodes may change over time due to current nodes disconnecting and new ones joining. Therefore, in the context of DRL for service orchestration in FC scenarios, designing a generalized policy that is able to handle the variable number of fog nodes is a crucial research challenge.

The main entities in the DRL-FORCH system architecture are illustrated in Figure 2. Our proposed DRL-FORCH orchestrator leverages retrieved telemetry data from the fog network to perform online decision-making. In the current implementation, FORCH employs Prometheus [22], paired with a simple Python-based system monitoring routine, to collect and store information on the resource utilization of the fog nodes, including usage data on CPU, RAM, disk, and network interfaces.

Formally, we consider an orchestrator managing a set V of fog nodes. Service requests from users arrive randomly, upon which the orchestrator chooses either to serve them in one of

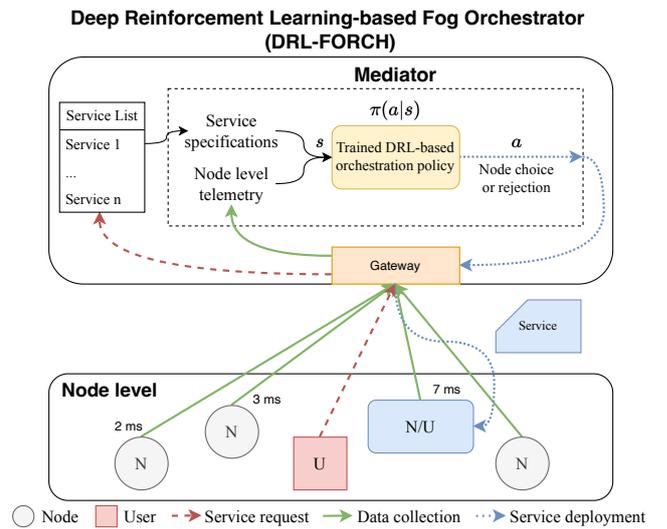


Fig. 2. Service deployment decision making in DRL-FORCH.

the available fog nodes or to reject them (e.g., by blocking the service request or by sending it to a cloud infrastructure). The FORCH system maintains an index of all the services it supports, including information needed to refer to each service in the orchestration context (name, identifier, service type), as well as to activate it (virtualization technology-specific details). Each request specifies the service the user needs, which is associated with a specific service type. In FORCH, services are classified into different types based on their deployment model, within the XaaS paradigm. Specifically, with references to the FORCH architecture introduced in [8] and extended in [23], we consider the following types of service:

- **Fog Virtualization Engine (FVE)**, instantiated according to the Infrastructure-as-a-Service (IaaS) model. It enables the deployment of a variety of computing environments on top of a generic virtualization layer available on a fog node (e.g., Docker).
- **Software Development Platform (SDP)**, instantiated according to the Platform-as-a-Service (PaaS) model. It allows the node to provide the user with a set of tools, including libraries, platforms, or interpreters to develop and run generic applications (e.g., Python SDK).
- **Application (APP)**, instantiated according to the Software-as-a-Service (SaaS) model. It lets a fog node host a specific application that can be accessed by consumers through a specific interface (e.g., a Web-based application).
- **Lightweight Atomic Functions (LAF)**, instantiated according to the Function-as-a-Service (FaaS) model. Based on the event-driven serverless computing execution model, this service is fully managed by the fog node and activated at need (e.g., real-time multimedia transcoding).

We presume that nodes that make their resources available to the fog orchestration system support at least one of the services that FORCH recognizes. Nodes offering SDP, APP,

and/or LAF services will only be able to be employed to serve requests for the specific type of service they offer. Conversely, we expect that nodes able to instantiate FVE services (i.e., nodes capable of deploying virtualized platforms) can instantiate any other type of services, provided a suitable source (e.g., a container image) exists for them.

Generally, we assume FaaS and SaaS elements to be the ones with tighter requirements on latency and softer needs in terms of computational power, while PaaS and IaaS elements to be less strict in terms of latency requirements but more demanding in terms of computational power. We argue that this is a sensible rationale, as the former two services types are generally associated with on-demand fast service provisioning, reflected in their looser requirements in terms of computational power, and their tighter requirements regarding latency times; on the opposite, the latter two service types usually correspond to a need for more computation power at the expense of service latency.

In our system model, we view fog nodes as heterogeneous devices with inherent limits on computational resources [24]. Specifically, we consider two classes of fog nodes: fixed nodes and mobile nodes. Fixed nodes are specialized nodes located at the edge of the local infrastructure, that assist the orchestrator whenever no suitable mobile node can be selected for the provisioning of services. As such, they possess vast computational resources, and provide services with lower latency times compared to the mobile ones. Mobile nodes, instead, represent devices that make their resources available to the orchestrator for a limited time, generally not known a priori, as they are reachable only when connected to the same local network of the orchestrator, before disconnecting, either logically or physically. As such, those nodes are likely to offer fewer computational resources than fixed nodes, and are generally expected to exhibit less predictable latency performance. However, mobile nodes are expected to provide valuable support in handling services that have softer constraints in terms of latency and required computational power, allowing the orchestrator to reserve fixed nodes for more demanding tasks, thereby improving resource usage efficiency.

In this work, we assume that resource occupation in terms of CPU, RAM, disk, and bandwidth of fog nodes is represented as normalized values in $[0, 1]$. We assume that fog nodes are available if their resource occupation does not exceed 0.95 and 0.5 for fixed and mobile nodes, respectively, i.e., we assume that mobile nodes have approximately half the overall capacity of fixed nodes in terms of CPU, RAM, disk, and bandwidth.

Furthermore, we make some assumptions regarding the service latency that each fog node can provide and those required by each service class. In this work, we assume that each service belongs to a “latency class” that goes from 1 to 10, where class 1 includes services with the tightest requirements on latency, and 10 denotes best-effort. Each fog node in the system is also associated with a latency class, representing the lowest latency value the node can support (i.e., the services it can deploy satisfying their requirements on latency). Specifically, we randomly assign a latency class

in the range from 5 to 10 for mobile nodes and from 1 to 5 for fixed ones, respectively, consistently with the above rationale. This information is leveraged in the process of choosing on which node to activate a requested service.

Finally, we assume that, for each service FORCH supports, upper bounds on the requested computational resources are known a priori by the orchestrator.

B. Formulating Orchestration in Fog Computing as an MDP

To train a DRL-based orchestrator for FC networks, we must provide an MDP formulation for our FC environment. In particular, we need to provide suitable definitions for the state space \mathcal{S} , the action space \mathcal{A} , and the reward function R . Crucially, we need to engineer our state and action spaces to be compatible with our Deep Sets-based permutation-equivariant policy and permutation-invariant value function neural networks.

Action Space: we consider a discrete action space of dimension $|V| + 1$. Given a service request, the orchestrator either allocates it to one of the $|V|$ fog nodes or rejects it. Rejection can be due to a lack of computational resources in the fog nodes, or proactive, i.e., rejecting a service request with the long-term goal of serving more in the future. Rejection can be intended as offloading the service request to the cloud, in which the availability of computational resources is assumed to be unconstrained.

We note that the action space grows linearly with $|V|$, which may pose scalability issues for large numbers of fog nodes. Still, not all actions may be simultaneously permitted given a specific state. Indeed, a fog node may not have enough computational resources to satisfy a service request or may not host the appropriate containerized service. A simple solution found in previous literature [9], [10] is to issue negative rewards (i.e., penalties) whenever an invalid action is chosen. Even though penalties discourage the agent from choosing invalid actions, it is well-known that in DRL sparse (i.e., infrequent), large rewards are detrimental to convergence. As invalid actions are a priori known by the orchestrator, we leverage *invalid action masking* in policy gradient algorithms, which has been found to yield significantly better performance and sample efficiency than invalid action penalties [25]–[27]. Action masking works by setting the log-probabilities of invalid actions to $-\infty$ before sampling an action, according to a state-dependant action mask. As such, we ensure that our orchestrator never chooses invalid actions. Formally, we define the action mask of node $i \in \{0, \dots, |V|\}$ at state s as follows:

$$\text{mask}(s)[i] = \begin{cases} \text{true} & \text{if } i \text{ has enough resources and hosts} \\ & \text{the appropriate service container} \\ \text{false} & \text{otherwise} \end{cases} \quad (3)$$

Whereas for action $|V| + 1$, i.e., rejection, the action mask is always set to *true*. The possibility of rejection ensures that in all states, at least one action is permitted. In this way, we exclude the possibility that the agent may “lock itself” in states where all actions are invalid.

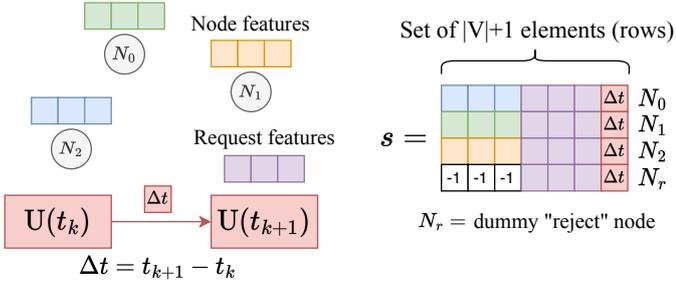


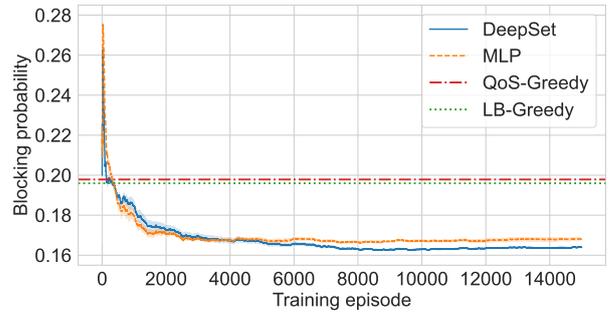
Fig. 3. Building a state for DRL-FORCH. The state s consists of a set of $|V| + 1$ elements, each element having a fixed number of features m .

State Space: we consider a state space of dimension $(|V|+1) \times m$, where m is the number of features per node. For each fog node, we choose the normalized CPU, RAM, disk, bandwidth utilization, and its measured delay as features. As our Deep Set-based policy returns a categorical probability distribution over the set elements (i.e., choosing a fog node), we must also introduce a “dummy” node representing the reject action. Therefore, we include an additional “reject” node with all features set to -1 . Furthermore, we consider request-specific features, namely, the CPU, RAM, disk, and bandwidth required by the current request. Finally, since the orchestrator is invoked at each service request arrival, the time interval Δt between two consecutive environment steps is not constant. Therefore, we also consider Δt as an input feature. The inclusion of Δt allows the agent to learn the traffic dynamics, as it explicitly conveys the inter-arrival time between subsequent requests, and implicitly the service duration time (by looking at the difference in resource occupation in fog nodes between two consecutive states). The request-specific features and Δt are replicated $n + 1$ times and concatenated to the node-specific features. Fig. 3 illustrates how the set-based state representation is built.

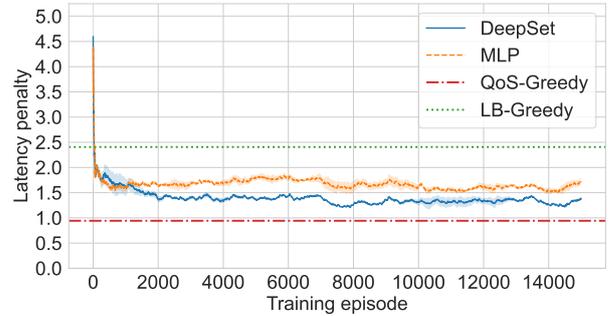
Reward Function: we consider a bi-objective optimization problem, such that i) the average blocking probability is minimized, and ii) the latency QoS requirements of each service are satisfied as much as possible. As such, upon either choosing a fog node or rejecting a service request, our reward function is shaped as follows:

$$r_t = \begin{cases} 1 & \text{if accepted and } L_{\text{node}} \leq L_{\text{service}} \\ -\frac{L_{\text{node}} - L_{\text{service}}}{L_{\text{node}}} & \text{if accepted and } L_{\text{node}} > L_{\text{service}} \\ -1 & \text{if rejected} \end{cases} \quad (4)$$

where L_{node} is the maximum latency class satisfiable by the chosen fog node, and L_{service} is the latency class required by the orchestrated service. When $L_{\text{node}} > L_{\text{service}}$, we define the latency penalty $L_{\text{pen}} = L_{\text{node}} - L_{\text{service}}$ as a measure of the QoS degradation. As such, the reward function is shaped such that it penalizes both i) blocking and ii) violation of latency service requirements. We note that, our reward shaping strategy is akin to assigning different objective priorities in multi-objective optimization, which depend on the specific application scenario [28]–[30]. In our case, we normalize the latency penalty such that the associated reward is always greater than -1 , meaning that rejecting a service request is



(a) Training blocking probability.



(b) Training latency penalty

Fig. 4. Training blocking probability and QoS metrics for PPO-DeepSet and the considered greedy heuristics and DRL-based baselines.

always considered more undesirable than accepting it with a degradation in the latency requirements.

We underline that objectives *i*) and *ii*) compete with each other. For instance, the blocking probability can be minimized by a deterministic load-balancing policy that distributes the requests evenly among all the available fog nodes, but this would very likely break the QoS requirements of the most latency-constrained service requests. Conversely, one may ensure QoS satisfaction by greedily filling the fixed nodes first, but failing to exploit the additional capacity provided by mobile nodes will lead to larger blocking probabilities. Therefore, learning an online orchestration policy that finds a good long-term trade-off between load balancing and latency constraints is not a trivial optimization task.

V. ILLUSTRATIVE NUMERICAL RESULTS

The goal of our illustrative numerical results is to empirically show that our Deep Set-based orchestrator can:

- 1) outperform standard greedy heuristic approaches in terms of QoS metrics optimization;
- 2) match or outperform a traditional MLP-based orchestrator with inference times up to an order of magnitude faster and lower memory occupation;
- 3) generalize its learned policy without the need for re-training (i.e., zero-shot generalization) to numbers of fog nodes up to two orders of magnitude larger than training.

In our simulation, for each service request, we assume the demanded computational resources (CPU, RAM, disk, and bandwidth) to be uniformly distributed in $[0.15, 0.3]$, $[0.1, 0.2]$,

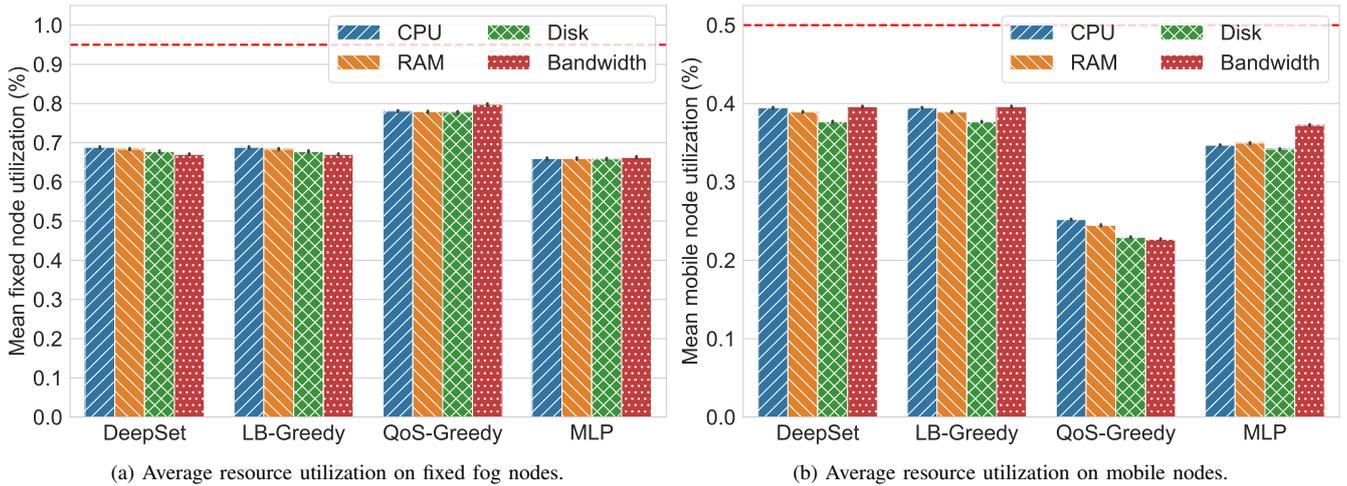


Fig. 5. Average resource utilization of fixed and mobile nodes for $|V| = 100$ fog nodes, mean arrival rate $\lambda = 1000$, and mean service duration $\mu = 1$. The dashed line indicates the 0.95 and 0.5 saturation thresholds for fixed and mobile fog nodes, respectively.

$[0.01, 0.1]$, and $[0.01, 0.02]$ for FVE, SDP, APP, and LAF service types, respectively. Similarly, we assume the requested latency classes to be uniformly distributed in $[4, 10]$, $[3, 7]$, $[1, 4]$, and $[1, 4]$ for FVE, SDP, APP, and LAF service types, respectively. Our training and evaluation protocol is structured as follows.

Training: we train on a small set of 10 fog nodes, with a fixed amount of offered traffic equal to 100 service requests. We used PPO for training our DRL-based policy. Specifically, we consider an episodic learning setting with episode length equal to 100, i.e., where the policy and the value function networks are updated every 100 request arrivals. Empirically, choosing an episode length equal to 100 provided the best trade-off between the convergence times of the policy network and the long-term optimization of our objective function. For implementing PPO, we borrowed from CleanRL [31] and Stable-Baselines3 [32]. Training times were in the order of ten minutes on a Macbook Pro M1 CPU.

Evaluation: we evaluate on environments with numbers of fog nodes $|V| = 50, 100, 500$ and 1000 , with a mean arrival rate $\lambda = 500, 1000, 5000$ and 10000 time-units, respectively, and a mean service duration $\mu = 1$ time-units for all cases. Thus, we evaluate zero-shot generalization of DS on instances up to two orders of magnitude larger than training.

Baselines: we compare our DS-based orchestrator with the following baseline orchestration policies:

- **LB-Greedy:** assigns the service request to the available node with the lowest current resource occupation. This policy aims at minimizing the blocking probability, at the expense of the latency penalties.
- **QoS-Greedy:** assigns the service request to the available node with the lowest measured service delay. This policy aims at minimizing the latency penalties, at the expense of the blocking probability.
- **MLP:** DRL-based orchestrator optimizing the same reward function as DS, but using an MLP neural network to

implement the policy and the value networks. In contrast to DS this baseline is evaluated on the same settings in which it was trained, due to the fixed input/output dimensionalities of MLPs.

We consider average blocking probability and latency penalty (as defined in (4) as performance metrics. For DS and MLP, numerical results are aggregated over five different random training seeds.

Numerical Results: Fig. 4 illustrates the evolution during training of blocking probability and latency penalty of DS and MLP, alongside the average values of QoS-Greedy and LB-Greedy. DS attains approximately 3% lower blocking probabilities than the greedy heuristic baselines, achieving a relative 15% improvement. Moreover, the average latency penalty of DS stands lower than 1.5, which is only 0.5 away from the QoS-Greedy policy. Overall, DS achieves the best trade-off between blocking probability and latency penalty, illustrating the capability of DRL to efficiently learn an effective online optimization strategy for both QoS metrics. Moreover, while DS and MLP learn similar policies, we observe that the policy learned by DS is slightly better than MLP for both the considered metrics. This result illustrates that the choice of a DS-based policy for orchestration in FC networks is indeed more desirable than a traditional MLP. Furthermore, other than outperforming an MLP, DS can be applied to an arbitrary number of fog nodes without the need for retraining.

Fig. 5 illustrates the resource occupation (CPU, RAM, disk, and bandwidth) of DS and the considered baselines in the training environment. We observe that, as expected, QoS-Greedy tends to favor fixed nodes, as they provide on average lower latency. Conversely, LB-Greedy tends to distribute the load between mobile and fixed nodes evenly, irrespective of the latency requirements. As illustrated in Fig. 4, both greedy approaches fail to provide a competitive blocking probability, as they do not take into account the dynamics of the traffic and cannot issue tactical proactive rejections. DS and MLP, on

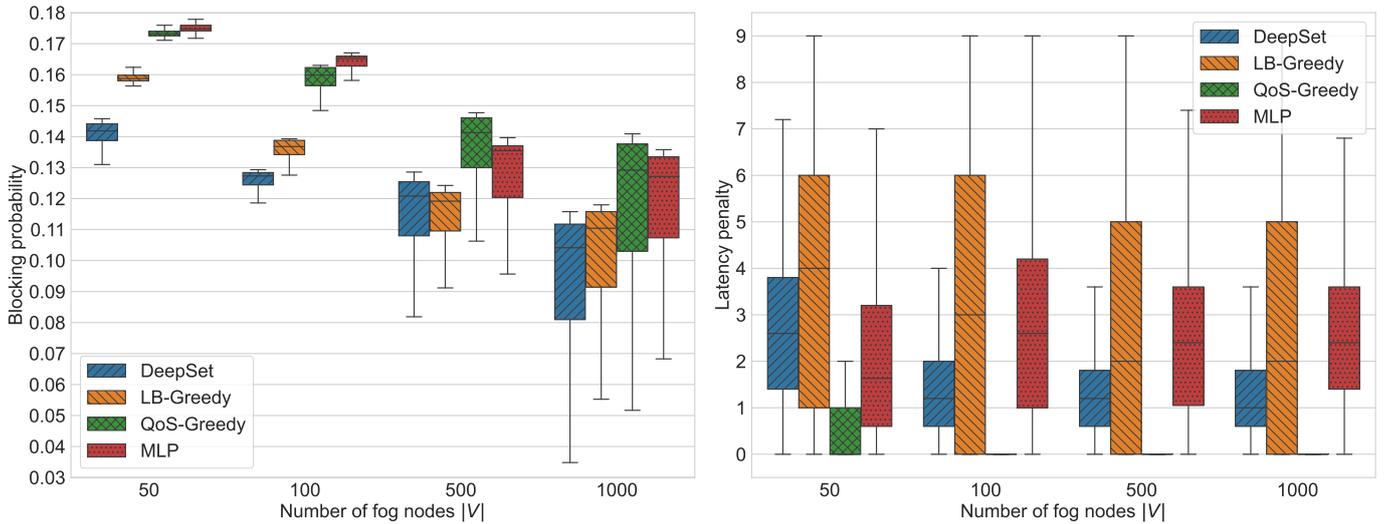


Fig. 6. Test blocking probabilities and latency penalties of DeepSet and the considered greedy heuristics and DRL-based baselines.

the other hand, make smart use of the available mobile nodes (without getting too close to overloading them), allowing them to achieve better resource utilization than both greedy heuristics. We note that the occupation of both fixed and mobile nodes is relatively far from their 0.95 and 0.50 threshold, respectively. This suggests that blocking mainly happens due to the unavailability of nodes hosting a specific containerized service rather than the unavailability of raw computational resources. This consideration reinforces the need for smart resource allocation policies in FC networks.

Fig. 6 illustrates the blocking probabilities and latency penalties of DS, MLP and the greedy baselines for test environments with $|V| = 50, 100, 500$, and 1000 fog nodes, with mean inter-arrival rates equal to $\lambda = 500, 1000, 5000$, and 10000 time-units, respectively, and service duration equal to $\mu = 1$ time-unit for all cases. We underline that for DS the same policy trained on $|V| = 10$ is reused in a zero-shot fashion, whereas for MLP a new policy must be trained from scratch due to the different dimensionalities of the input spaces, with training times spanning several hours for $|V| = 1000$. In particular, due to the growing complexity of the input space, MLP is unable to learn a competitive policy for large numbers of fog nodes. Conversely, DS extrapolates well its learned policy without the need for retraining, attaining the best trade-off between blocking probabilities and latency penalty up to $|V| = 1000$. In particular, up to $|V| = 100$, DS significantly outperforms all baselines in terms of blocking probability, and achieves latency penalties slightly worse than QoS-Greedy. For $|V| = 500, 1000$ LB-Greedy closes the gap with DS in terms of blocking probability, but DS achieves significantly lower latency penalties. We conclude that DS learned an effective and generalizable orchestration policy, achieving a smart trade-off between load balancing and satisfying QoS latency requirements.

Fig. 7 illustrates the blocking probability of DS and the other baselines for $|V| = 100$, $\mu = 1$ while varying λ . We can observe that both DS and MLP generalize well

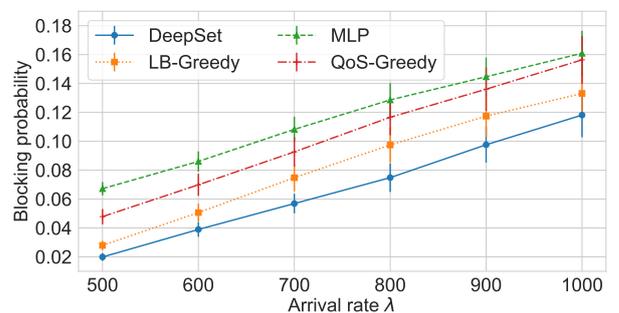


Fig. 7. Blocking probability of DeepSet and the considered baselines for $|V| = 100$ fog nodes, varying the service arrival rate λ .

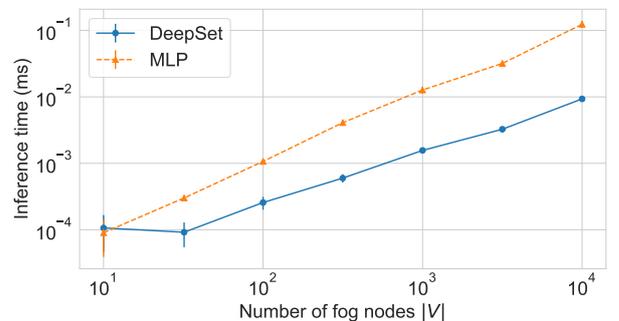


Fig. 8. Inference times (ms) of DeepSet and MLP with two hidden layers of size 512, for a varying number of fog nodes and 100 features per node.

with respect to the amount of offered traffic, with blocking probabilities scaling linearly as expected. Consistently with previous results, DS outperforms all other baselines in terms of blocking probability.

We conclude our scalability analysis by evaluating the inference times of the proposed Deep Set-based orchestrator for large numbers of fog nodes and features per node. Fig. 8 illustrates the inference times of DS-based and MLP-based policies with two hidden layers of dimension 512 and 100 features per fog node. Inference times were measured on a Macbook Pro M1 CPU. While DS and MLP show similar

inference times for smaller numbers of fog nodes, DS scales significantly better, with inference times approximately equal to 0.01ms for 10^4 fog nodes, an improvement of an order of magnitude compared to MLP. This is expected, as the size of MLP grows with the number of fog nodes, while the size of DS is independent of the number of fog nodes. As such, our DS-based policy provides excellent scalability in both time and memory complexity.

VI. CONCLUSION

In this work, we proposed DRL-FORCH, a Deep Reinforcement Learning-based orchestrator for Fog Computing networks built on top of the state-of-the-art FORCH orchestration system. To deal with the limitations of standard DRL-based solution for dealing with variable numbers of fog nodes, we engineered the online orchestration problem as a Machine Learning problem on sets, and we proposed the use of Deep Sets neural networks for implementing our DRL-based orchestration policy. Our illustrative numerical results demonstrate that our Deep Sets-based policy yields excellent zero-shot generalization for numbers of fog nodes up to two orders of magnitude larger than training, providing the best trade-off between blocking probability and satisfaction of service latency constraints. As future work, we plan on integrating a realistic fog node energy consumption model in the DRL reward function, and evaluating the performance of our DRL agent in large-scale smart city environments.

ACKNOWLEDGMENTS

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on “Telecommunications of the Future” (PE00000001 - program “RESTART”).

REFERENCES

- [1] F. Bonomi *et al.*, “Fog computing and its role in the internet of things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 13–16.
- [2] Z. M. Fadlullah *et al.*, “State-of-the-art deep learning: Evolving machine intelligence toward tomorrow’s intelligent network traffic control systems,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.
- [3] G. Stampa *et al.*, “A deep-reinforcement learning approach for software-defined networking routing optimization,” *arXiv preprint arXiv:1709.07080*, 2017.
- [4] N. Di Cicco *et al.*, “On deep reinforcement learning for static routing and wavelength assignment,” *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 28, no. 4: Mach. Learn. in Photon. Commun. and Meas. Syst., pp. 1–12, 2022.
- [5] X. Liu *et al.*, “Resource allocation for edge computing in iot networks via reinforcement learning,” in *ICC 2019-2019 IEEE international conference on communications (ICC)*. IEEE, 2019, pp. 1–6.
- [6] M. R. Raza *et al.*, “Reinforcement learning for slicing in a 5g flexible ran,” *J. Lightwave Tech.*, vol. 37, no. 20, pp. 5161–5169, 2019.
- [7] M. Zaheer *et al.*, “Deep sets,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS’17, 2017, p. 3394–3404.
- [8] G. Davoli *et al.*, “A fog computing orchestrator architecture with service model awareness,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2131–2147, 2022.
- [9] H. Sami *et al.*, “Demand-driven deep reinforcement learning for scalable fog and service placement,” *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2671–2684, 2022.
- [10] A. Nassar and Y. Yilmaz, “Deep reinforcement learning for adaptive network slicing in 5g for intelligent vehicular systems and smart cities,” *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 222–235, 2022.
- [11] Y. Sun *et al.*, “Deep reinforcement learning-based mode selection and resource management for green fog radio access networks,” *IEEE IoT Journal*, vol. 6, no. 2, pp. 1960–1971, 2018.
- [12] J. C. Guevara *et al.*, “Qos-aware task scheduling based on reinforcement learning for the cloud-fog continuum,” in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 2328–2333.
- [13] Y. Wei *et al.*, “Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor-critic deep reinforcement learning,” *IEEE IoT Journal*, vol. 6, no. 2, pp. 2061–2073, 2018.
- [14] Y. Wang *et al.*, “Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications,” *IEEE Trans. Ind. Infor.*, vol. 15, no. 2, pp. 976–986, 2018.
- [15] Z. Wei *et al.*, “Dynamic many-to-many task offloading in vehicular fog computing: A multi-agent drl approach,” in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 6301–6306.
- [16] Z. Gao *et al.*, “Fast adaptive task offloading and resource allocation via multi-agent reinforcement learning in heterogeneous vehicular fog computing,” *IEEE Internet of Things Journal*, 2022.
- [17] S. N. Afrasiabi *et al.*, “Reinforcement learning-based optimization framework for application component migration in nfv cloud-fog environments,” *IEEE Transactions on Network and Service Management*, 2022.
- [18] Q. Chang *et al.*, “Cooperative edge caching via multi agent reinforcement learning in fog radio access networks,” in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 3641–3646.
- [19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.
- [20] J. Schulman *et al.*, “Proximal policy optimization algorithms,” 2017. [Online]. Available: <https://arxiv.org/abs/1707.06347>
- [21] J. Gilmer *et al.*, “Neural message passing for quantum chemistry,” in *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, 2017.
- [22] Prometheus. [Online]. Available: <https://prometheus.io/>
- [23] G. F. Pittalà *et al.*, “Function-as-a-service orchestration in fog computing environments,” in *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE, 2022, pp. 364–366.
- [24] B. Costa *et al.*, “Orchestration in fog computing: A comprehensive survey,” *ACM Computing Surveys*, vol. 55, no. 2, pp. 1–34, 2022.
- [25] O. Vinyals *et al.*, “Starcraft ii: A new challenge for reinforcement learning,” 2017. [Online]. Available: <https://arxiv.org/abs/1708.04782>
- [26] S. Huang and S. Ontañón, “A closer look at invalid action masking in policy gradient algorithms,” in *Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022*, 2022.
- [27] J. W. Nevin *et al.*, “Techniques for applying reinforcement learning to routing and wavelength assignment problems in optical fiber communication networks,” *J. Opt. Commun. Netw.*, vol. 14, no. 9, pp. 733–748, Sep 2022.
- [28] N. Di Cicco *et al.*, “Optimization over time of reliable 5g-ran with network function migrations,” *Computer Networks*, vol. 215, p. 109216, 2022.
- [29] N. H. Thanh *et al.*, “Energy-aware service function chain embedding in edge-cloud environments for iot applications,” *IEEE IoT Journal*, vol. 8, no. 17, pp. 13 465–13 486, 2021.
- [30] G. Sun *et al.*, “Energy-efficient provisioning for service function chains to support delay-sensitive applications in network function virtualization,” *IEEE IoT Journal*, vol. 7, no. 7, pp. 6116–6131, 2020.
- [31] S. Huang *et al.*, “Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms,” *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022.
- [32] A. Raffin *et al.*, “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

5.2. Reinforcement Learning for Multi-Objective Service Orchestration

In the previous paper, we considered a multi-objective service orchestration problem jointly optimizing blocking probability and service latency. However, this problem was dealt with by merging the two objectives into one, and then by solving the problem as a single-objective problem. In practice, it is complex, if not impossible, to predict a priori how objectives will interact, and it is, therefore, challenging to decide on one single preference between the objectives. Moreover, preferences between objectives might evolve with time, requiring finding a new objective combination and fitting a new model from scratch. In this paper, we propose adopting a proper multi-objective approach to RL for service orchestration, with the goal of approximating the complete Pareto Front of policies. In this way, we can separate optimization (i.e., computing a Pareto Front) from decision-making (i.e., choosing a trade-off from the Pareto Front), allowing the network manager to make informed decisions once presented with all possible trade-offs. To explore interesting trade-offs in service orchestration in a realistic scenario, we consider the devices' energy consumption as an objective, and we build a realistic energy consumption model as a function of computational load based on profiling of real-world edge devices. Numerical results show that our RL algorithm successfully identifies multiple trade-off points where significant energy savings (up to 48%) can be achieved with minimal reduction in the service acceptance probabilities, and outperforms conventional heuristics by 30% in terms of expected costs.

Scalable and Energy-Efficient Service Orchestration in the Edge-Cloud Continuum with Multi-Objective Reinforcement Learning

Nicola Di Cicco, *Graduate Student Member, IEEE*, Gaetano Francesco Pittalà, *Graduate Student Member, IEEE*, Gianluca Davoli, *Member, IEEE*, Davide Borsatti, *Member, IEEE*, Walter Cerroni, *Senior Member, IEEE*, Carla Raffaelli, *Senior Member, IEEE*, Massimo Tornatore, *Fellow, IEEE*

Abstract—The Edge-Cloud Continuum represents a paradigm shift in distributed computing, seamlessly integrating resources from cloud data centers to edge devices. However, orchestrating services across this heterogeneous landscape poses significant challenges, as it requires finding a delicate balance between different (and competing) objectives, including service acceptance probability, offered Quality-of-Service, and network energy consumption. To address this challenge, we propose leveraging Multi-Objective Reinforcement Learning (MORL) to approximate the full Pareto Front of service orchestration policies. In contrast to conventional solutions based on single-objective RL, a MORL approach allows a network operator to inspect all possible “optimal” trade-offs, and then decide a posteriori on the orchestration policy that best satisfies the system’s operational requirements. Specifically, we first conduct an extensive measurement study to accurately model the energy consumption of heterogeneous edge devices and servers under various workloads, alongside the resource consumption of popular cloud services. Then, we develop a set-based MORL policy for service orchestration that can adapt to arbitrary network topologies without the need for retraining. Illustrative numerical results against objective-centric heuristics show that our MORL policy outperforms baselines by 30% on average over a broad set of objective preferences, and generalizes to network topologies up to 5x larger than training.

Index Terms—Service Orchestration, Edge-Cloud Continuum, Multi-Objective Reinforcement Learning, Energy Profiling

I. INTRODUCTION

In the era of pervasive computing, the integration of Edge and Cloud resources forms a crucial continuum that enables the provisioning of scalable and efficient services [1]. This Edge-Cloud Continuum (ECC) leverages the strengths of both Edge and Cloud environments to support services requiring low latency and high computational power. Given the multitude of involved network segments and technologies, as well as the dynamic nature of resource allocation, orchestrating services across the ECC presents significant challenges, particularly in terms of scalability and energy efficiency [2]. Indeed, managing heterogeneous resources and ensuring optimal performance across distributed environments necessitates advanced solutions that can intelligently navigate

the trade-offs between resource optimization and network energy consumption.

In this context, Deep Reinforcement Learning (DRL) has emerged as a promising solution for solving service orchestration, thanks to its flexibility and effectiveness in solving general sequential decision-making problems [3]. However, conventional DRL-based approaches [4], [5] for service orchestration suffer from two major shortcomings: i) they must be retrained from scratch if the network changes in size, e.g., if the number of edge nodes increases, and ii) they either consider only one objective, or multiple objectives are merged into a single one via, e.g., hand-picked linear combinations. This results in DRL orchestration policies that are fixed to one specific deployment scenario (i.e., the training scenario) and one specific trade-off between the objectives [6]. This greatly limits the applicability of DRL in real-world service orchestration, since the number of edge devices in the network might increase, and the orchestrator’s preferences between heterogeneous goals (e.g., service latency and network energy consumption) might either evolve with time or be outright unknown at training time. Additionally, to effectively train a model for minimizing energy consumption, it is essential to quantify the energy demands of different devices, as well as how different workloads impact energy usage.

To address the aforementioned challenge, we propose solving service orchestration leveraging Multi-Objective Reinforcement Learning (MORL). Specifically, we consider three competing objectives, namely, service acceptance probability, service latency, and network energy consumption, assuming that the orchestrator’s preferences among them are a priori unknown. Then, we leverage concepts from MORL to approximate the Pareto Front of policies, i.e., to derive a (small) set of candidate policies, each one achieving a locally optimal trade-off. All of these policies are simultaneously available to the orchestrator, which can choose at any time the policy to deploy according to its current preferences. Moreover, we leverage an ad-hoc neural network based on Deep Sets [7], enabling scalability to network sizes several times larger than training without the need for retraining. We build the proposed framework upon our prior work [8], retaining the same approach to service provisioning, but generalizing it for broader adoption of the Everything-as-a-Service (XaaS) model over the ECC. Every time the activation of a new service is requested, a DRL-based policy is employed to select the best

N. Di Cicco and M. Tornatore are with the Department of Electronics, Information and Bioengineering (DEIB), Politecnico di Milano, Italy. E-mail: {nicola.dicicco, massimo.tornatore}@polimi.it.

G. F. Pittalà, G. Davoli, D. Borsatti, W. Cerroni, and C. Raffaelli are with the Department of Electrical, Electronic, and Information Engineering (DEI) “G. Marconi”, University of Bologna, Italy. E-mail: {francesco.pittalà, gianluca.davoli, davide.borsatti, walter.cerroni, carla.raffaelli}@unibo.it.

computing resource(s) to serve the request, based on available monitoring data and service specifications.

The novel contributions to this work are as follows: (i) we perform extensive profiling and service micro-benchmarking for measuring the energy and resource consumption of different edge devices across a large set of realistic XaaS services. This allows us to incorporate a realistic model of energy consumption in the training of a DRL agent; (ii) we explicitly formulate service orchestration as a MORL problem, applying a custom “outer loop” training algorithm, and (iii) we perform extensive numerical evaluation, and empirically verify that our learned MORL policies find better trade-offs than baseline heuristics, and that their strong performance generalizes to networks up to 50x larger than the one seen during training.

Our dataset, comprising profiling of the energy consumption of edge devices and micro-benchmarks of popular XaaS services, is publicly available, and our code is open-sourced.¹

The remainder of the paper is structured as follows. In Section II, we outline the scenario, the orchestration system’s mechanics, and the XaaS principles and service models. In Section III, we define our service orchestration problem as a MORL issue, starting with MORL basics and detailing our problem formulation. In Section IV, we present our evaluation of service computing resource utilization within the given scenario. In Section V, we validate our approach with numerical experiments, first analyzing service acceptance and energy consumption, then including the QoS objective. In Section VI, we briefly survey related work, highlighting our novel contributions. Finally, we outline conclusions and future research directions in Section VII.

II. ORCHESTRATION IN THE EDGE-CLOUD CONTINUUM

This Section provides an overview of the scenario under consideration and details how it is modeled for our service provisioning purposes. Additionally, it describes the internal mechanics of the orchestration system, including the logical elements and functional interactions between such elements and the users. Finally, it addresses the principles of XaaS by elaborating on the considered service models.

A. Reference Edge-Cloud scenario

The scenario we consider for the design and evaluation of our proposed solution is illustrated in Fig. 1, and it comprises the following elements:

- Edge servers (referred to as *servers*): devices located within the same premises as base station equipment. They possess decent computational power and storage, enabling the possibility to run virtualized appliances and services for end users on them. To model their energy consumption, we choose small computing platforms such as Intel NUCs and Shuttle MiniPCs as reference.
- Far-Edge devices (referred to as *nodes*): they include sensors, boards, or mobile phones. They may join the edge cluster temporarily and share their computing capabilities [9]. They are battery-powered and possess limited

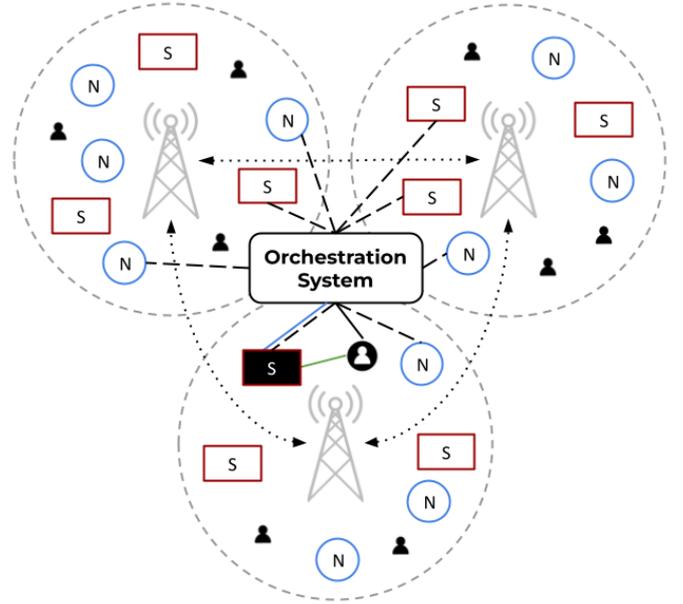


Fig. 1. Reference scenario featuring edge cells containing computing resources, represented by blocks labeled S for servers and N for nodes, which are managed by the Orchestration System. Users are depicted by small human-like icons. Solid lines indicate the interactions involved in service requests, deployment, utilization, and monitoring. A server and a user are highlighted to denote their involvement in the example service deployment illustrated by the sketched interactions, consistently with Fig. 2.

computational capabilities. To model their energy consumption, we choose low-power devices such as Raspberry Pis as reference.

- Orchestration System (referred to as *orchestrator*): the entity in charge of providing services through the employment of computational capabilities of both servers and nodes. The orchestrator is the one who collects telemetry data from the resources and makes decisions on which one to utilize for the provisioning of services.
- Users: agents that connect to the edge cluster and request the provisioning of services.

In this scenario, each time a user submits a service request, the orchestrator attempts to fulfill it by managing resources within the coverage areas of various base stations, encompassing both wired servers and wireless nodes. We assume that each user can be allocated a service on any edge resource via the shortest path established through the connections between the base stations. The orchestration functionality we focus on is the selection of the most appropriate computing resources to fulfill the service request. Upon reception of the request, the orchestrator selects the optimal node to provide the service and attempts to deploy the service on it. Once the deployment is complete, the system responds to the user with an endpoint that they can use to reach the deployed service.

B. Orchestration architecture

The logical structure of the orchestration system is depicted in Fig. 2. The orchestrator is divided into two layers: the *Service layer* and the *Resource layer*. The former operates at a higher level of abstraction and comprises functionalities

¹Data and code will be made public in case of acceptance of this paper.

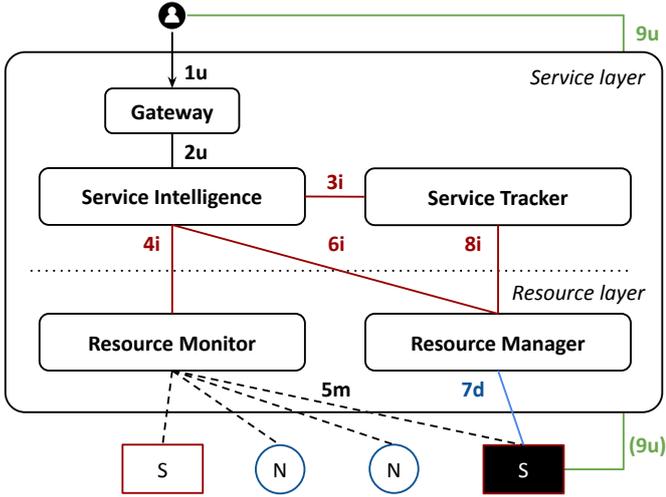


Fig. 2. Abstracted representation of the orchestration system, with its functional elements and logical connections. The interactions have different line styles and labels based on the type of communication (i for internal, u for users, s for signaling, d for deployment) and a number to indicate the sequence of these interactions.

such as service placement and Quality of Service (QoS) enforcement. The latter focuses on the practical aspects of deploying service components onto available resources and monitoring their performance and availability. These layers, in turn, consist of multiple functional components, that we briefly describe in the following.

- **SERVICE INTELLIGENCE:** it uses the information collected from nodes and users to make decisions on the optimal placement of services that users request.
- **SERVICE TRACKER:** it tracks the services currently provided by the nodes. The orchestrator updates this database upon deployment confirmation and removes services no longer in use based on information gathered during the monitoring phase.
- **RESOURCE MONITOR:** it monitors the resources of the various nodes interconnected within the system. It collects essential metrics such as CPU, RAM, Disk, and power consumption, enabling the orchestrator to make informed decisions about which node to use to serve incoming requests.
- **RESOURCE MANAGER:** it manages the resources under the control of the orchestrator. It can communicate with different operating systems and deploy services on various virtualization platforms (e.g., Docker).

To facilitate the understanding of the mechanisms involved in the orchestration environment, Fig. 2 illustrates the communications taking place within the orchestrator system when a service request is submitted to the framework. A user initiates the service provisioning process by submitting a request for a service to the orchestrator, through its gateway endpoint (1u). The request is routed to the SERVICE INTELLIGENCE element, which is responsible for making the decision on which node should be employed to serve the request (2u). To do so, the SERVICE INTELLIGENCE element contacts the SERVICE TRACKER to obtain information about which

services are currently being provided within the system and which of these are already available on the nodes (3i). It then requests monitoring information to the RESOURCE MONITOR, obtaining the current resource status of the nodes (4i). The RESOURCE MONITOR element, in turn, queries the various interconnected nodes to obtain their available resources (CPU, RAM, disk, network load) at that particular moment (5m). At this point, the SERVICE INTELLIGENCE has all the necessary information to decide which node is the best fit to serve the newly received request. It then communicates this decision to the RESOURCE MANAGER (6i). The RESOURCE MANAGER directly interacts with the selected node to deploy the service for the user's request if it is not already present on the node; otherwise, it reserves an instance for the end user (7d). After deploying the service, the RESOURCE MANAGER may inform the SERVICE TRACKER about the new service deployed on the node (8i). The user can then access the service on the computing resource it has been deployed to (9u).

C. Service Models

The establishment of the ECC as a new distributed scenario for service provisioning has facilitated the diffusion of traditional Cloud Computing XaaS service models, i.e., Function-as-a-Service (FaaS), Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) categories, beyond their original boundaries. The same categories can be now adopted in different network segments, including in resource-constrained distributed environments.

The SaaS model allows access to a software application instance running on a remote server, which can be extended to the ECC, providing minimal service activation time with data exchange limited to input parameters and output results. The PaaS model allows consumers to access a remote platform for developing custom applications using provided programming languages, libraries, and tools, extending to the ECC where consumers can deploy source code to be compiled and executed on the edge node. The IaaS model offers consumers virtualized computing, storage, and networking resources to run custom systems, applicable to the ECC where services are deployed on edge nodes through compatible virtualization environments, with potential additional steps and data exchanges for service activation.

Notably, extending these Cloud service models to the Edge-to-Cloud Continuum retains similar characteristics, with increasing generality and flexibility from SaaS to PaaS to IaaS, while service activation times increase. Thus, an effective orchestration system must understand how services can be deployed across different models and nodes, balancing flexibility and activation times.

By incorporating service model awareness and employing the previously described systematic flow, the system ensures an efficient and effective service placement process within the reference scenario, optimizing resource utilization and service delivery across the available resources.

III. MULTI-OBJECTIVE RL FOR SERVICE ORCHESTRATION

In this Section, we formalize our service orchestration problem in the context of MORL. Specifically, we first provide

a brief background on MORL, discussing key definitions and foundational concepts. Then, we describe our considered service orchestration MORL problem.

A. Multi-Objective Reinforcement Learning

MORL problems are formalized as Multi-Objective Markov Decision Processes (MOMDPs). Formally, a MOMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, T, \gamma, \mu, \mathbb{R} \rangle$ where \mathcal{S} is the state space, \mathcal{A} is the action space, $T: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the probabilistic transition function, $\gamma \in (0, 1]$ is the discount factor, $\mu: \mathcal{S} \rightarrow [0, 1]$ is the initial state's distribution, and $\mathbb{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}^d$ is a vector-valued reward function, having one reward for each of the d objectives. In our specific problem, a state $s \in \mathcal{S}$ is the information available to the orchestrator at the decision time, and an action $a \in \mathcal{A}$ encodes on which node the agent decided to provision a service request.

We model our service orchestration agent as a policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$ that outputs an action given a state as an input. We assume that π lies in a policy space Π , which comprises all of the deterministic and stochastic policies.² Given a MOMDP, we define the vector-valued value function of a policy π as

$$\mathbf{V}^\pi = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_{t+1} \mid \pi, \mu \right], \quad (1)$$

where \mathbf{r}_t is the reward value at the t -th step. In other words, the vector-valued value function of a policy in a MOMDP is the expected discounted return for each of the d objectives.

In the literature, it is often the case that multiple objectives are *scalarized* into a single value, based on some user-defined notion of utility for each objective. Formally, we define a *utility function* $u: \mathbb{R}^d \rightarrow \mathbb{R}$ to be a map of the d objectives into a single scalar objective. In general, u can be any monotonic (possibly non-linear) function of the objectives.

In this paper, we consider a scenario where the utility function is a priori unknown. In other words, we assume that we do not know how to express relative preferences between the objectives such that we achieve an ‘‘optimal’’ trade-off. In MORL, in contrast to single-objective Reinforcement Learning (RL), there might exist multiple optimal policies, each one achieving the optimum for some utility function (i.e., achieving a different, but ‘‘locally optimal’’ trade-off). In the case of monotonic utility functions, this set of optimal policies is the Pareto Front (PF). Formally, we define the PF as

$$\text{PF}(\Pi) = \{\pi \in \Pi \mid \mathbf{V}^\pi \succ_P \mathbf{V}^{\pi'} \forall \pi' \in \Pi\}, \quad (2)$$

where \succ_P is the Pareto dominance relation, defined as

$$\mathbf{V}^\pi \succ_P \mathbf{V}^{\pi'} \iff (\mathbf{V}_i^\pi \geq \mathbf{V}_i^{\pi'} \forall i) \wedge (\exists i: \mathbf{V}_i^\pi > \mathbf{V}_i^{\pi'}), \quad (3)$$

where \mathbf{V}_i^π indicates the i -th objective of value function \mathbf{V}^π . In other words, a policy π Pareto-dominates a policy π' if π performs better than π' over at least one objective, and does not perform worse in any of the other objectives. The PF contains the policies whose value functions Pareto-dominate the ones of all the other possible policies. Note that, there

might exist multiple policies in the PF yielding the same value function. For this reason, we are generally not interested in reconstructing the whole PF, but only the smallest possible set containing policies whose value functions lie in the PF. In this context, we can define a Pareto Coverage Set (PCS) as

$$\text{PCS}(\Pi) = \{\pi \in \text{PF}(\Pi) \mid \forall \pi' \in \Pi, \exists \pi: \mathbf{V}^\pi \succ_P \mathbf{V}^{\pi'}\}. \quad (4)$$

In other words, a PCS contains policies whose value vectors are Pareto-optimal. Note that, in contrast to the PF, a PCS can contain only one policy for each Pareto-optimal value function.

We now restrict our attention to linear utility functions, i.e., functions of the form $u(\mathbf{V}^\pi) = \mathbf{w}^T \mathbf{V}^\pi$, where $\mathbf{w} \in \mathbb{R}^d$ is a vector of weights for each of the d objectives. Formally, we define the Convex Hull (CH) as

$$\text{CH}(\Pi) = \{\pi \in \Pi \mid \exists \mathbf{w}: \mathbf{w}^T \mathbf{V}^\pi \geq \mathbf{w}^T \mathbf{V}^{\pi'} \forall \pi' \in \Pi\}, \quad (5)$$

which is the set of policies achieving the optimal objective value for some linear utility function. Again, there might exist many redundant policies on the CH achieving the same value function. For this reason, similarly to the PCS, we can define a Convex Coverage Set (CCS) as

$$\begin{aligned} \text{CCS}(\Pi) &= \\ &= \{\pi \in \text{CH}(\Pi) \mid \forall \mathbf{w}, \exists \pi: \mathbf{w}^T \mathbf{V}^\pi \geq \mathbf{w}^T \mathbf{V}^{\pi'} \forall \pi' \in \Pi\}. \end{aligned} \quad (6)$$

A particularly important result [10] states that, if we allow stochastic policies, a CCS is sufficient for reconstructing the PCS. Specifically, we can reconstruct the PCS by taking mixture distribution policies from the CCS. Therefore, even though a CCS contains the set of optimal policies under linear scalarizations, we can use it to derive optimal policies under non-linear scalarizations as well. For this reason, MORL algorithms under a priori unknown utilities seek to construct an approximation of the smallest-possible CCS, from which the PCS can be then derived [11].

B. An Outer Loop Algorithm for MORL

Having established that the CCS is sufficient for reconstructing the whole PF, we need to design an algorithm for approximating the CCS. Specifically, our goal is to design a *multi-policy* algorithm, such that we return a different policy depending on a user-specified preference vector \mathbf{w} .

Popular multi-policy MORL algorithms run an ‘‘outer loop’’ over a single-objective DRL algorithm. In a nutshell, outer loop algorithms iterate over many possible scalarizations of the objective function, and for each one, they train a single-objective DRL policy. The critical design choice in these algorithms lies in how to choose the scalarization (e.g., by grid search, or with a runtime exploration strategy), and in how to reuse information between different training runs.

In this regard, we experimented with several state-of-the-art algorithms with a reference implementation in MORL-Baselines [12], such as Envelope Q-Learning [13], GPI-PD [14], [15]. However, for our specific MORL environment, we found it simpler and more effective to run a dense grid search over the space of possible weight vector and run the customized single-objective PPO algorithm [16] we introduced

²A deterministic policy maps a state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$. A stochastic policy maps a state $s \in \mathcal{S}$ to a *distribution* over all possible actions in \mathcal{A} .

in [8]. Specifically, our PPO algorithm incorporates two fundamental design elements:

- **Deep Sets Policy.** Our algorithm uses Deep Sets [7] to implement the policy, as opposed to conventional Multi-Layer Perceptrons. The core idea is to treat the state representation of the environment as a *variable-sized set of elements* (i.e., a set of edge nodes), as opposed to a fixed-size vector [8]. This allows our policy to support variable-sized observation and action spaces (i.e., it can be deployed in networks of variable size without retraining). Moreover, the computational complexity of the Deep Sets network not only scales linearly with the input size, but can take advantage of massive parallelization, since the input set's elements can be processed in parallel.
- **Action Masking.** We employ action masking [17]–[19] such that the policy, both during training time and inference time, is never allowed to select an invalid action (i.e., provisioning on a node that does not have sufficient capacity, or provisioning an AMD64-only service on an ARM64 node). This greatly improves the training time of the algorithm as opposed to, e.g., issuing a large negative reward in correspondence to invalid actions.

These features are critical to the performance, generalization, and training-time efficiency of our algorithm [8]. Thanks to these enhancements, our PPO algorithm is able to converge in a few minutes, which enables us to run a dense grid search over preference vectors \mathbf{w} . We then extract the CCS by filtering out non-dominated policies according to Eq. (3). Since PPO learns stochastic policies, we are able to approximate a PCS given a CCS by taking mixtures of CCS policies.

C. Service Orchestration

We assume that our MORL policy π runs in the Orchestration System, and thus has a global view of the resource occupation of each device in the ECC. At each service request arrival, the policy is tasked to decide whether to admit or not the service request and, in the case of admission, to decide on which device to allocate the request. We assume that the policy operates in a purely online fashion, that is, decisions are taken one at a time as service requests arrive, and that the policy doesn't have any prior knowledge of future demands. Our MORL problem comprises three distinct objectives, which we describe as follows.

Acceptance Probability. The first objective is the maximization of the total *service acceptance probability*, which we impose with the following reward:

$$r_{t+1}^{\text{acc}} = \begin{cases} 1 & \text{if accepted,} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Recall from Eq. (1) that the vector-valued value function \mathbf{V}^π of a policy π consists of the accumulation over the time of rewards. Therefore, the summation of the above reward over a time horizon will yield the total number of accepted requests.

Quality-of-Service. The second objective is the maximization of *Total Quality-of-Service*, which we define in terms of

service latency. Specifically, we impose minimization of the total service latency with the following reward function:

$$r_{t+1}^{\text{QoS}} = \begin{cases} -d(\text{Service}_t, a_t) & \text{if accepted,} \\ 0 & \text{otherwise,} \end{cases} \quad (8)$$

where d is a distance function between a user and the edge device. In our case, as illustrated in 1, we consider the shortest-path distance through the base stations' network between the user and its associated edge device. As long as we assume that the latency is a strictly monotonic function of the distance, the exact expression of the latency does not change the Pareto dominance relationship between solutions. Similarly to before, the value function for this reward will consist of the total QoS penalty for all the accepted requests. Note that, there is a trade-off between the service acceptance rate and the latency; for instance, to minimize latency, a naive policy could simply reject requests whose closest node is currently congested.

Energy Consumption. The final objective is the minimization of the *total energy consumption*. Specifically, we assume that the orchestrator *does not know* the exact power consumption profile of the servers and edge devices, and thus has to learn it throughout the training process. With reference to the power consumption profiles illustrated in Table II, we compute the energy consumption reward (in Joules) as the total energy dissipated between two consecutive time steps, i.e., between two consecutive service request arrivals. Formally, if we have N devices, we express the total energy consumption between time-steps t and $t + 1$ as

$$r_{t+1}^{\text{energy}} = \sum_{i=1}^N \int_t^{t+1} P_i(\text{CPU}_i(t)) dt, \quad (9)$$

where $P_k(\cdot)$ is the instantaneous power consumption of the i -th device as a function of its CPU load, and $\text{CPU}_i(t)$ is the time-evolving CPU occupation of the i -th device. In our specific problem, the CPU occupation might decrease between time steps t and $t+1$ due to one or multiple service departures. Summing Eq. (9) over a time horizon will yield the total energy consumption over that time horizon.

Reward Shaping for Energy Consumption. We empirically verified that our DRL agent struggled to learn good energy-efficient orchestration policies with the above reward function. This is because the reward value is largely influenced by the environment's dynamics (i.e., energy consumption and service departures in nodes other than the chosen one), which, from the point of view of the agent, act as confounders. To this end, we adopt *reward shaping*, that is, feed the agent a reward that is correlated with the "real" reward, and is at the same time more informative (in our case, less noisy). This introduces a bias in the learning process, at the advantage of more consistent convergence. Specifically, we define our shaped reward as the difference in instantaneous power in the chosen node before and after service provisioning, that is

$$r_{t+1}^{\text{QoS}} = \begin{cases} P_i(\text{CPU}_i(t)) - P_i(\text{CPU}_i(t+1)) & \text{if accepted,} \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

Similarly to before, there is a clear trade-off between acceptance probability and energy consumption. For instance, rejecting all requests trivially minimizes energy consumption.

Action Space. We consider a discrete action space of dimension $|V| + 1$, where $|V|$ is the number of edge nodes, representing that a service request is provisioned in node $1 \dots |V|$ or rejected.

State Space. In line with our prior work [8], we adopt a set-based representation of dimension $(|V| + 1, m)$, where $|V| + 1$ is the number of elements (i.e., edge nodes) in the input set, and m is the number of per-node features. Each node is characterized by its CPU and disk occupations, and by one-hot-encoded device type. Moreover, for each node, we concatenate the following service-level information: the required CPU and disk, the user’s position coordinates, the service type, the device type, and the time interval elapsed between the previous request’s arrival and the current one. Finally, we include a dummy “reject” node with all features set to -1 . This set-based representation, combined with Deep Sets networks, allows us to support an action space of the same dimensionality of the number of set elements (i.e., $|V| + 1$), without having to fix the value of $|V|$.

IV. SERVICE MICROBENCHMARKS

In this Section, we detail our assessment of computing resource usage for services within the scenario described in Section II-A. We examine the relationship between deployed service, CPU usage, and energy consumption across a range of devices, from low to high power, providing insights into how computational demands influence power efficiency. Through systematic monitoring and analysis, we evaluate the impact of service provisioning on both performance and energy usage, enabling informed decision-making for orchestration processes. We consider the employed computing resources to be CPU-bound, which is reasonable considering the characteristics of the involved elements, and the fact that the dynamic nature of service provisioning primarily strains computing power rather than RAM or disk usage. Consequently, CPU usage plays a central role in our evaluations.

CPU Scaling Factors. In our measurement campaign, we profile the CPU occupation of services on a single device for each computing architecture, namely AMD (x86) and ARM. For simulation purposes, we then adapt these measurements to other devices within the same category, referencing CPU benchmark ratings from publicly available sources.³ Specifically, we compute a set of *scaling factors* as the ratio between the ratings of the additional devices and those of the benchmarked ones. Based on this approach, we consider a Raspberry Pi 3B+ to have 1.42x the computing power of a Raspberry Pi 2B and 0.45x the computing power of a Raspberry Pi 4B. Similarly, we consider a NUC to have 0.98x the computing power of a Shuttle. This scaling approach allows us to dimension the CPU resource consumption in heterogeneous devices in a reasonably accurate manner, ensuring a reliable performance evaluation across different hardware configurations.

³<https://www.cpubenchmark.net/>

A. Resource usage

To validate our approach, we ran an extensive data collection regarding the characterization of both edge devices and services that we took as a reference to our simulation scenario. Based on the categories described in Section II-A, we define two categories for edge devices: nodes and servers. Nodes are typically represented by boards with limited computational power, storage, and RAM, primarily featuring ARM-based CPUs and powered by batteries. We consider these characteristics to represent mobile devices connected to our edge cluster, offering a limited set of resources while remaining within the boundaries of the simulated edge network. Servers, on the other hand, are represented by fixed equipment installed on the same premises as base stations, providing greater computational power and stability. This categorization is consistent with the one proposed in [20].

To begin with, we formalize a list of services that reflect a real-world ECC scenario, addressing multiple applications with varying requirements in terms of CPU, RAM, and disk. Keeping the XaaS paradigm in mind, we also define different provisioning mechanisms for the services in our environment, consistently with the models described in Section II-C. Specifically, we consider a hierarchy of service deployment models, from IaaS (the most generic, heaviest one), to PaaS, to SaaS, to FaaS (the most specific, lightest one). We assume each model can be deployed on top of another one, provided that the “base” service comes before the other one in the hierarchy. For instance, a service deployed as FaaS might be installed on top of a compatible service deployed as PaaS. A fundamental distinction between services lies in whether they can operate natively on a server or node, or if they require virtualization. For virtualization, we employ the Docker container runtime environment, compiling a list of Docker images from the official repository⁴ to enable the execution of services in a virtualized form. We then evaluate the performance of all services (including native and virtualized) under realistic customer use cases.

For this assessment, we run these services on an Intel NUC device (more details on it in Section IV-B) and use a bash script to evaluate CPU usage as measured by the `top` utility. This script launches the service and records the CPU usage over 100 iterations, with 1-second intervals between iterations. We then compute the mean across all iterations, for each service. The resulting values are reported in Tables I and II, and the meaning of the columns is described in the following.

- *ID*: a numeric identifier for the service.
- *Name*: a descriptive service identifier.
- *Type*: the category the service falls into, according to the XaaS paradigm, i.e., FaaS, SaaS, PaaS, or IaaS.
- *Image*: the Docker image needed to deploy the service.
- *Size*: the size (disk occupation) of the image.
- *Idle*: the idle CPU usage of the deployed service.
- *Load*: the CPU usage of the service while in execution.
- *Base Service*: an additional service that needs to be deployed for this service to work (e.g., a FaaS service might require Python to be executed).

⁴<https://hub.docker.com/>

TABLE I
EXCERPT OF THE AMD64 SERVICE LIST CREATED BASED ON THE MEASUREMENT CAMPAIGN CONDUCTED ON INTEL NUC7I7DNKE

ID	Service Name	Type	Image	Size [MB]	Idle [%]	Load [%]	Base Service	Setup Time [s]
2	Python	PaaS	python	1010	0.176	15.22	-	39
11	Discord	SaaS	kasmweb/discord	2570	2.643	18.36	-	220
10	Matlab	PaaS	mathworks/matlab	5600	0.168	12.34	-	220
20	SHA256 Integrity Check	FaaS	-	-	-	12.54	Python	-
30	Ubuntu	IaaS	ubuntu	78	0.171	9.832	-	5

TABLE II
EXCERPT OF THE ARM64 SERVICE LIST CREATED BASED ON THE MEASUREMENT CAMPAIGN CONDUCTED ON RASPBERRY PI 3B+

ID	Service Name	Type	Image	Size [MB]	Idle [%]	Load [%]	Base Service	Setup Time [s]
4	Python	PaaS	python	1022	1.183	30.32	-	188
5	VLC	SaaS	kasmweb/vlc	2466	3.417	82.33	-	566
7	Gimp	PaaS	kasmweb/gimp	2440	3.512	30.86	-	571
9	Ubuntu	IaaS	ubuntu	101	1.077	4.172	-	18
13	SHA256 Integrity Check	FaaS	-	-	-	79.30	Python	-

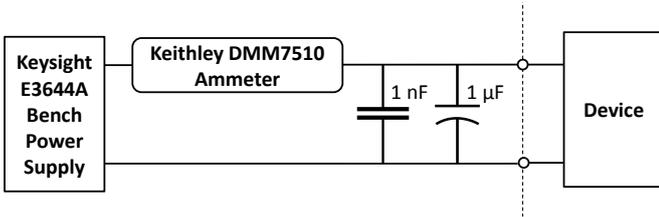


Fig. 3. Test bench configuration for the power absorption measurement.

- *Setup Time*: the time required to download the required image and instantiate the container.

As the table contains details of heterogeneous services, some columns do not apply to all entries and are, therefore, left empty. The *Base Service* column is relevant only to services deployed on top of another service, as explained previously in this Section. For instance, the *SHA256 Integrity Check* service, deployed as FaaS, relies on the *Python* service, deployed as PaaS, as its base. The integrity check is implemented as a function within this base service. Consequently, it does not make sense to associate an image and idle resource usage specifically to the *SHA256 Integrity Check* service, as these details are relevant to the base service only.

B. Energy consumption

We profile the energy consumption of a range of devices representative of real-world scenarios, from low-power Single-Board Computers (SBCs) to high-power desktop-like computers, which are comparable in performance to computing elements typically found across the ECC. Such devices are listed in Table III.

To measure the energy consumption of the devices consistently, we set up a test bench comprising a bench power supply (Keysight Technologies E3644A) and a multimeter configured as an ammeter (Keithley DMM7510), connected as shown in Fig. 3. This configuration enables precise measurement of the current drawn by each device under test. To compensate for power fluctuations, we add two capacitors (1 nF tantalum and 1 μ F electrolytic) in parallel to the power input of the device. The smaller capacitor filters out high-frequency noise, while

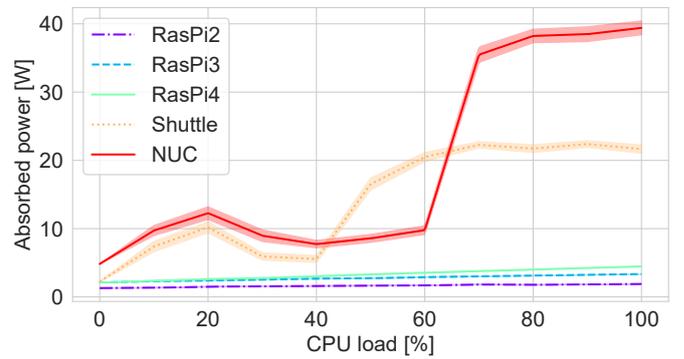


Fig. 4. Absorbed power as a function of CPU load, showing the significant difference in energy consumption of various edge computing devices.

the larger one helps cope with the higher current absorption some devices exhibit at boot. This dual-capacitor setup ensures that the measurements are not affected by transient spikes or dips, providing more accurate and reliable data on the energy consumption of the devices at different CPU loads. To generate such loads, the devices are subjected to synthetic computational tasks that utilize the CPU in increments from 0% to 100%, increasing by 10% every 10 seconds.

Fig. 4 illustrates the measured power consumption profiles, over a range of CPU usage amounts. As expected, devices with lower computational power also draw less current. An interesting observation, however, is that energy consumption does not always increase linearly with CPU load. While Raspberry Pis exhibit a linear relationship, MiniPCs demonstrate a step-like pattern between load and absorbed current. These devices draw significantly more power under higher loads compared to lower loads. This finding is consistent with the experimental results reported in [21]. Moreover, as shown in the figure, we observe a dip in power consumption in the range between 30% and 60% CPU load, which may seem counterintuitive. This phenomenon can be attributed to several factors related to how modern processors manage power and performance, such as Dynamic Voltage and Frequency Scaling (DVFS) [22] and advanced power management techniques [23], dynamically adjusting the voltage and frequency

TABLE III
DEVICES EMPLOYED FOR ENERGY PROFILING, AND THEIR MAIN HARDWARE CHARACTERISTICS.

Name	Description	CPU	RAM [GB]	Disk [GB]	Scaling Factor
RasPi2	RaspberryPi SBC, model 2B	4-core ARM Cortex-A7	1	16	1.42 (ARM)
RasPi3	RaspberryPi SBC, model 3B+	4-core ARMv8 Cortex-A53	1	16	1 (ARM)
RasPi4	RaspberryPi SBC, model 4B	4-core ARMv8 Cortex-A72	4	16	0.45 (ARM)
Shuttle	Shuttle MiniPC	4-core 10th-gen Intel i5	16	250	0.98 (AMD)
NUC	Intel NUC 7I7DNKE MiniPC	4-core 8th-gen Intel i7	16	500	1 (AMD)

of a CPU based on the workload. At lower CPU loads, the system might lower the CPU frequency and voltage to save power, leading to decreased energy consumption. Initially, as the CPU load increases, the frequency and voltage are increased to meet the performance requirements, resulting in higher energy consumption. However, at certain intermediate loads (like between 30% and 60%), the CPU might not need to run at full frequency, allowing it to operate more efficiently at a lower frequency, which could result in reduced power consumption. Additionally, modern CPUs have various efficiency states where they can operate at different performance and power levels. These states are optimized to provide the best performance per watt for specific load conditions. Between 30% and 60% CPU usage, the CPU might be operating in a more efficient state, where it balances performance and power consumption optimally, resulting in lower energy usage compared to slightly higher or lower loads.

V. ILLUSTRATIVE NUMERICAL RESULTS

In this Section, we validate the performance of our proposed approach via an extensive set of numerical experiments. For visualization purposes, we first analyze a simplified MORL problem considering only the service acceptance probability and energy consumption objectives. Then, we evaluate the complete problem, comprising also the latency objective.

Simulation Settings. With reference to the scenario illustrated in Fig. 1, we consider a square area of 1km^2 and we generate base station locations according to a Poisson Point Process (PPP) with $\lambda_{\text{PPP}} = 50$. This base station density is in line with that of the city of Bologna, which, according to the latest measurement [24], had a rough density of 35 base stations per km^2 in its city center in 2022. For the training scenario, we consider 20 edge nodes uniformly distributed according to the device types illustrated in Table III. We assume that the service requests' arrival rate is Poisson and that their duration is exponential. To account for service durations depending on the service type, we assume that PaaS, SaaS and IaaS requests last on average 2x, 3x and 4x more than FaaS, respectively. For testing, we increase the number of edge nodes to 50, 100, and 200. We dimension the requests' arrival rate and service duration such that a load-balancing policy results in a service acceptance rate above 90%.

Baselines. Throughout this Section, we will compare our proposed solution against the following non-learning-based heuristics. The following heuristics have been designed to approximately cover all extreme points in the objective space (i.e., low blocking, low latency, and low energy consumption), without explicitly addressing all objectives simultaneously.

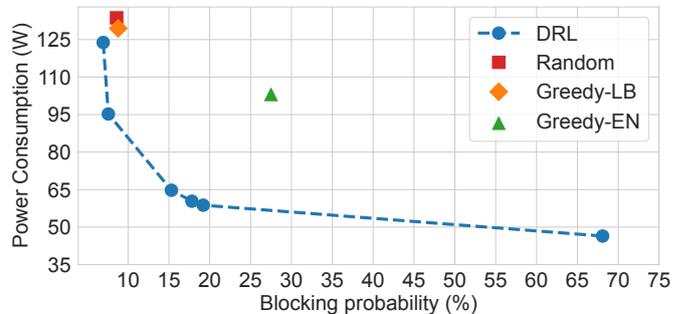


Fig. 5. PF approximation derived by DRL for blocking probability vs. energy consumption, alongside objective-driven baselines.

- **Random:** selects randomly one among the nodes that have sufficient resources for service provisioning.
- **Greedy Load Balancing (Greedy-LB):** selects the node that has the lowest occupation among CPU and disk.
- **Greedy Low Latency (Greedy-Lat):** selects the closest (in terms of distance) among the admissible nodes.
- **Greedy Low Energy (Greedy-En):** if ARM64 nodes are available, it selects the least loaded one in terms of CPU. Otherwise, it selects one random AMD64 node, but only if its CPU occupation is less than 50%.

Performance Metrics. We quantitatively evaluate the performance of a MORL policy in terms of Expected Cost, which we define as the average scalarized objective value obtained by the algorithm over randomly sampled objective weights. This metric quantifies how well a policy performs in the presence of novel preferences. In our case, since we assume that the orchestrator's preferences are a priori unknown, we uniformly sample objective weights in the d -dimensional unit simplex, where d is the number of objectives. Besides this, we visually inspect the resulting PFs and qualitatively assess trade-offs. For visual clarity, we normalize with respect to the mean of the Random baseline.

A. Acceptance Rate vs. Energy Consumption

We now focus on the trade-offs between energy consumption and blocking probability. Fig. 5 illustrates the PF approximated via DRL, and points corresponding to the Random, Greedy-LB and Greedy-En heuristics. We observe that the policies discovered by DRL dominate the heuristics, but also DRL identifies a clear elbow point. Specifically, we can reduce the network energy consumption by 48% if we degrade the service acceptance probability by approximately 10%, after which we start experiencing diminishing returns. Moreover, we note that when DRL exclusively focuses on minimizing

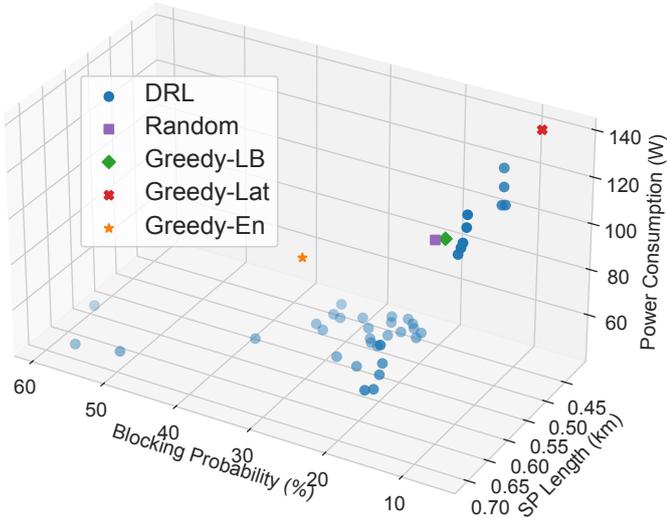


Fig. 6. 3D PF approximation for blocking probability vs. latency vs. energy consumption, alongside objective-driven baselines.

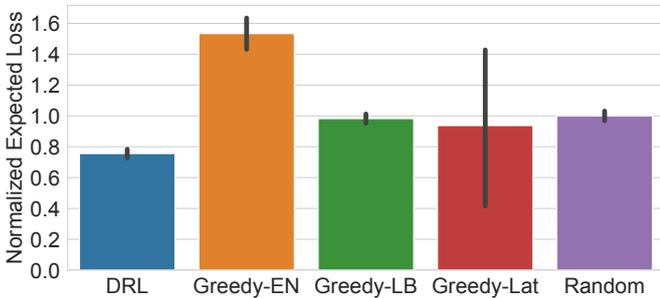


Fig. 7. Expected cost of blocking rate vs. latency vs. energy consumption with respect to randomly-sampled weight vectors.

blocking, it performs similarly to Greedy-LB and Random (with DRL being marginally better). Instead, by degrading the acceptance probability by 1.5%, DRL discovers significantly better trade-offs between the blocking probability and energy consumption compared to all the baselines. This is because DRL learns by experience to tactically exploit the low-power consumption regions of AMD64 devices, balancing resource utilization and low energy consumption.

B. Acceptance Rate vs. Energy Consumption vs. Latency

We now study the complete problem considering also service latency as an optimization objective.

Fig. 6 illustrates a projection of the resulting three-dimensional objective space. Specifically, alongside the blocking probability and the power consumption, we illustrate the average length of the shortest path (SP) between each provisioned request and its associated edge node. Comparing the blocking rate and the energy consumption, we observe the same trend previously noted in Fig. 5. In terms of latency, the Greedy-LB heuristic shows the lowest average SP length at a low blocking probability, but this comes at the cost of the highest total power consumption. Instead, our algorithm discovers a dense region of “middle-ground” policies achieving similar shortest-path lengths, but with 57% less power consumption, albeit at the price of a 10% in blocking. We

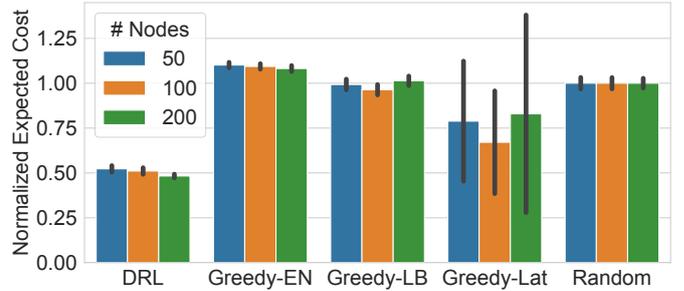


Fig. 8. Expected cost of blocking rate vs. latency vs. energy consumption with respect to randomly-sampled weight vectors. The DRL policy is trained on a network with 20 edge nodes, and then the number of available edge nodes is increased without retraining.

note that we are not advocating for a specific trade-off, rather we are highlighting the capability of a MORL algorithm to uncover a diverse set of candidate solutions.

Fig. 7 illustrates the expected loss of DRL and the baselines, averaged over 1000 random weight vectors, and normalized with respect to the Random policy. We observe that the DRL policy outperforms the baselines by 30% on average. This is to be expected, as the baselines are tailored for optimizing one specific objective, while our DRL multi-policy algorithm can adapt to many different preferences from the orchestrator. Specifically, Greedy-EN is strongly penalized as it overly focuses on minimizing energy consumption, disregarding the other objectives. Greedy-Lat, though performing well in terms of latency and blocking, strongly underperforms in terms of energy consumption, thus resulting in a high-cost variance. This result highlights the advantages of treating service orchestration as a MORL problem, since the PF allows us to quantitatively assess how competing objectives interact and to make informed decisions on the system’s trade-offs.

C. Scalability Analysis

We conclude our performance evaluation by assessing the generalizability of our learned DRL policies to testing environments larger than those used for training. Specifically, we test the DRL policies described in Section V-B without retraining on three test scenarios, which include 50, 100, and 500 edge nodes, respectively.

Fig. 8 illustrates the resulting expected costs. Indeed, we note that the trends previously observed in Fig. 7 do not change with the problem size, with DRL outperforming the baselines by 30% on average. We remark that choosing Deep Sets as neural networks allows us to apply the policy under variable input and output sizes. This is in contrast with conventional neural network models, whose input and output space are fixed by their architecture, and changing them requires re-training.

VI. RELATED WORK

The adoption of Machine Learning (ML) techniques, particularly DRL, has substantially improved service placement strategies within the edge-cloud continuum, enhancing resource utilization and minimizing latency. However, the adoption of these strategies may be limited by a lack of comprehensive simulation tools that can evaluate performance at the edge

with the same depth and accuracy typically achieved in cloud environments. This section presents an overview of machine learning advancements in service placement within the edge-cloud continuum, also highlighting relevant simulation tools.

The work in [2] explores intelligent service orchestration within edge cloud networks, emphasizing adaptive service management techniques that respond dynamically to network conditions to optimize resource utilization and service quality. Distributed machine learning techniques are used in [25] to optimize resource sharing among multiple users in mobile edge computing systems, enhancing overall computational efficiency and system performance. Additionally, the work in [26] discusses dynamic task offloading techniques in mobile-edge computing networks, focusing on minimizing completion time and energy consumption through actor-critic learning structures, which optimize resource allocation among mobile devices. In [27], ML approaches are employed for task and resource allocation in mobile-edge computing-based networks, illustrating the role of edge computing in reducing latency for Internet of Things (IoT) devices. In [28], the integration of ML into network management and orchestration showcases the potential of adaptive resource management in edge-based networking paradigms. This approach aids in the dynamic orchestration of network resources, enhancing service delivery and network efficiency. Also, the authors in [5] and [29] highlight the critical role of ML in dynamic resource allocation and task offloading, using DRL to manage mobile edge computing environments effectively under various network conditions. The research in [6] delves into computational resource allocation using DRL in low latency edge computing networks, aimed at supporting communications with finite blocklength codes and enhancing communication responsiveness. Additionally, DRL is leveraged in [30] to optimize task scheduling within IoT frameworks, highlighting strategies to enhance system reliability and reduce response times, crucial for IoT applications.

The importance of realistic simulations for validating ML-based service placement strategies is evident in [31] and [32]. These studies provide insights into the application of ML algorithms in practical scenarios, showing their effectiveness in reducing latency and improving service delivery. The simulator in [33] facilitates the modeling of both computational and networking resources, enabling the evaluation of various architectures and the impact of system performance factors such as server capacity and mobility. The study in [34] addresses the computation offloading challenge in distributed edge computing networks through a DRL-based offloading algorithm, which considers real-time network states and task attributes to optimize server load distribution across heterogeneous networks. The research in [35] presents a proactive service placement and migration model based on Markov Decision Process (MDP), tailored to adapt service deployment dynamically to fluctuating user demands and server proximity. This strategic positioning helps minimize service latency and improve system efficiency.

Collectively, these studies underscore the pivotal role of advanced ML techniques in managing the complexities of service placement within the ECC, enhancing the scalability

and operational efficiency of networked systems in diverse environments.

The problem of multi-objective optimization for service provisioning at the Edge has been addressed in the literature with both learning- and non-learning-based approaches. The typical target is finding a trade-off between various conflicting metrics such as energy consumption, task delay, operational cost, and service availability. In [36], a multi-objective evolutionary algorithm was proposed to minimize energy consumption and task delay in Mobile Edge Computing (MEC) systems, showing significant improvements over benchmark solutions and extending network lifetime. Similarly, [37] introduced a multi-objective clustering evolutionary algorithm to optimize cost and energy consumption while meeting deadline constraints, demonstrating superior performance in multi-objective optimization tasks. Addressing task scheduling, [38] proposed an algorithm that combines Pareto-archived evolution strategy with nondominated sorting methods to minimize execution latency and energy consumption, achieving better results compared to other algorithms. In IoT and Edge Computing environments, [39] tackled the optimization of service placement and load distribution using a multi-objective genetic algorithm, effectively balancing operational cost and service availability. Moreover, [40] presented an improved MORL approach for minimizing application completion time, energy consumption, and usage charges, outperforming other methods across several metrics. Lastly, [4] and [41] explored MORL-based solutions for task allocation and offloading in fog and MEC systems, respectively, demonstrating their effectiveness in achieving Pareto-optimal solutions and optimizing resource utilization in dynamic environments. These studies collectively highlight the potential of multi-objective algorithms to enhance service provisioning at the Edge by effectively balancing multiple performance metrics.

VII. CONCLUSION

Our work demonstrates that Multi-Objective Reinforcement Learning (MORL) is a valuable tool for service orchestration, effectively discovering optimal trade-offs between service request blocking probability, latency, and energy consumption. For example, our algorithm identifies an elbow point where significant energy savings (up to 48%) can be achieved with only a minor reduction in acceptance probability. Compared to objective-centric heuristics, our MORL policy consistently outperforms the baselines, showing an average improvement of 30% and scaling to topologies five times larger than the training set. This makes our approach both practical and efficient for managing dynamic network environments.

Future work could explore the integration of more sophisticated energy consumption models to better characterize workloads and enhance energy efficiency optimization. Additionally, implementing and testing the MORL policy in real-world network deployments will help validate its practical benefits and uncover potential challenges in live scenarios. Another promising direction is the integration with digital twin technology to accurately characterize service response time, providing a virtual environment to simulate and optimize real-time service orchestration.

ACKNOWLEDGMENT

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on “Telecommunications of the Future” (PE00000001 - program “RESTART”) and on “Ecosystem for Sustainable Transition in Emilia-Romagna” (ECS00000033 - program “ECOSISTER”).

REFERENCES

- [1] L. Bittencourt *et al.*, “The internet of things, fog and cloud continuum: Integration and challenges,” *Internet of Things*, 2018.
- [2] M. Johnson *et al.*, “Intelligent service orchestration in edge cloud networks,” *IEEE Network*, 2020.
- [3] Z. Zhong *et al.*, “Machine learning-based orchestration of containers: A taxonomy and future directions,” *ACM Comput. Surv.*, vol. 54, no. 10s, sep 2022. [Online]. Available: <https://doi.org/10.1145/3510415>
- [4] M. A. Ibrahim and S. Askar, “An intelligent scheduling strategy in fog computing system based on multi-objective deep reinforcement learning algorithm,” *IEEE Access*, 2023.
- [5] A. Kumar *et al.*, “Resource allocation with edge computing in iot networks via machine learning,” *IEEE Internet of Things Journal*, 2020.
- [6] L. Yang *et al.*, “Deep reinforcement learning for resource allocation in low latency edge computing networks,” *IEEE Communications Letters*, 2020.
- [7] M. Zaheer *et al.*, “Deep sets,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS’17, 2017.
- [8] N. Di Cicco *et al.*, “Drl-forch: A scalable deep reinforcement learning-based fog computing orchestrator,” in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE, 2023.
- [9] R. Tourani *et al.*, “Democratizing the edge: A pervasive edge computing framework,” *arXiv preprint arXiv:2007.00641*, 2020.
- [10] P. Vamplew *et al.*, “Constructing stochastic mixture policies for episodic multiobjective reinforcement learning tasks,” in *AI 2009: Advances in Artificial Intelligence: 22nd Australasian Joint Conference, Melbourne, Australia, December 1-4, 2009. Proceedings 22*. Springer, 2009.
- [11] C. F. Hayes *et al.*, “A practical guide to multi-objective reinforcement learning and planning,” *Autonomous Agents and Multi-Agent Systems*, Apr. 2022.
- [12] F. Felten *et al.*, “A toolkit for reliable benchmarking and research in multi-objective reinforcement learning,” in *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023.
- [13] R. Yang *et al.*, *A generalized algorithm for multi-objective reinforcement learning and policy adaptation*. Red Hook, NY, USA: Curran Associates Inc., 2019.
- [14] L. N. Alegre *et al.*, “Sample-efficient multi-objective learning via generalized policy improvement prioritization,” in *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS ’23. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2023.
- [15] T. Basaklar *et al.*, “PD-MORL: Preference-driven multi-objective reinforcement learning algorithm,” in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: <https://openreview.net/forum?id=zS9sRyaPFIJ>
- [16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [17] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vechnyevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, “Starcraft ii: A new challenge for reinforcement learning,” *arXiv preprint arXiv:1708.04782*, 2017.
- [18] S. Huang *et al.*, “Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms,” *Journal of Machine Learning Research*, 2022.
- [19] J. Santos *et al.*, “Efficient microservice deployment in kubernetes multi-clusters through reinforcement learning,” in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024.
- [20] Linux foundation whitepaper on Sharpening the edge ii: Diving deeper into the If edge taxonomy and projects. [Online]. Available: <https://lfdge.org/resources/whitepapers-reports/>
- [21] A. Bellin *et al.*, “A preliminary study on the power consumption of virtualized edge 5g core networks,” in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE, 2023.
- [22] J. Mao *et al.*, “Modeling energy consumption of virtual machines in DVFS-enabled cloud data centers,” in *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2020.
- [23] A. Katal *et al.*, “Energy efficiency in cloud computing data centers: a survey on software technologies,” *Cluster Computing*, 2023.
- [24] Opendata municipality of Bologna - Radio devices list. Accessed: July 27, 2024. [Online]. Available: <https://opendata.comune.bologna.it/explore/dataset/elenco-impianti-antenne-telefonia/>
- [25] Y. Zhang *et al.*, “Distributed machine learning for multiuser mobile edge computing systems,” *IEEE Transactions on Vehicular Technology*, 2020.
- [26] X. Fang *et al.*, “Dynamic task offloading and resource allocation for mobile-edge computing,” *IEEE Transactions on Mobile Computing*, 2020.
- [27] Y. Liu *et al.*, “A machine learning approach for task and resource allocation in mobile-edge computing-based networks,” *IEEE Access*, 2020.
- [28] L. Zhao *et al.*, “When machine learning meets network management and orchestration in edge-based networking paradigms,” *IEEE Communications Magazine*, 2020.
- [29] X. Chen *et al.*, “Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa,” *IEEE Transactions on Network Science and Engineering*, 2020.
- [30] P. Sharma *et al.*, “Deep reinforcement learning based mobile edge computing for iot,” *IEEE Internet of Things Journal*, 2020.
- [31] J. Lee *et al.*, “Deep reinforcement learning-based task scheduling in iot edge computing,” *IEEE Transactions on Mobile Computing*, 2020.
- [32] D. Wang *et al.*, “A cooperative computation offloading strategy with on-demand deployment of multi-uavs in uav-aided mobile edge computing,” in *Proceedings of the IEEE International Conference on Computer Communications*. IEEE, 2020.
- [33] C. Sonmez *et al.*, “Edgecloudsim: An environment for performance evaluation of edge computing systems,” *Transactions on Emerging Telecommunication Technologies*, 2018.
- [34] Z. Li *et al.*, “Distributed edge computing offloading algorithm based on deep reinforcement learning,” *IEEE Internet of Things Journal*, 2020.
- [35] K. Narendra *et al.*, “Learning-based microservice placement and migration for multi-access edge computing,” in *Proceedings of the IEEE International Conference on Cloud Computing*. IEEE, 2020.
- [36] A. Bozorgchenani *et al.*, “Multi-objective computation sharing in energy and delay constrained mobile edge computing environments,” *IEEE Transactions on Mobile Computing*, 2021.
- [37] L. Pan *et al.*, “A multi-objective clustering evolutionary algorithm for multi-workflow computation offloading in mobile edge computing,” *IEEE Transactions on Cloud Computing*, 2023.
- [38] J. Li *et al.*, “Multiobjective oriented task scheduling in heterogeneous mobile edge computing networks,” *IEEE Transactions on Vehicular Technology*, 2022.
- [39] A. M. Maia *et al.*, “An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing,” *Computer Networks*, 2021.
- [40] F. Song *et al.*, “Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach,” *Future Generation Computer Systems*, 2022.
- [41] N. Yang *et al.*, “Multi-objective deep reinforcement learning for mobile edge computing,” in *2023 21st International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2023.

6 | Continual In-Network Learning

ML has become a ubiquitous technology in communications networks, to the point that the research community is investigating solutions for deploying trained ML models in programmable Application-Specific Integrated Circuits (ASICs), such as P4 switches [175]. However, it is well-known that the lifecycle of a ML model does not end at training, but instead a model will require continuous updates. This is because, as the original training data becomes outdated over time, the deployed ML model will suffer from continuing performance degradation. To solve this fundamental problem, in the following extended abstract we propose an architecture for continual updates of in-network ML models. Specifically, we propose integrating Active Learning with Continual Learning for optimally updating a trained ML model with the least amount of annotated data. Specifically, implementing Active Learning in the data plane allows the ML model to act not only as a traffic classifier, but also as a selective packet mirror for forwarding to the control plane novel traffic patterns to be used for model updates. Our preliminary numerical results on a representative dataset illustrate that our proposed architecture can achieve similar performance to an oracle model by learning incrementally and with a fraction of the total annotated data. To further make a case for continual adaptation, we plan to extend this preliminary study with additional use cases and datasets and investigate deployment on real programmable switches.

Poster: Continual Network Learning

Nicola Di Cicco
Politecnico di Milano
Milan, Italy

Andrea Melis
Università di Bologna
Bologna, Italy

Amir Al Sadi
Università di Bologna
Bologna, Italy

Gianni Antichi
Politecnico di Milano
Milan, Italy

Chiara Grasselli
Università di Bologna
Bologna, Italy

Massimo Tornatore
Politecnico di Milano
Milan, Italy

ABSTRACT

We make a case for in-network Continual Learning as a solution for seamless adaptation to evolving network conditions without forgetting past experiences. We propose implementing Active Learning-based selective data filtering in the data plane, allowing for data-efficient continual updates. We explore relevant challenges and propose future research directions.

CCS CONCEPTS

• **Networks** → **In-network processing**; *Programmable networks*;
• **Computing methodologies** → **Active learning settings**; **Online learning settings**.

KEYWORDS

In-network Machine Learning, Continual Learning, Active Learning, Programmable Data Planes

ACM Reference Format:

Nicola Di Cicco, Amir Al Sadi, Chiara Grasselli, Andrea Melis, Gianni Antichi, and Massimo Tornatore. 2023. Poster: Continual Network Learning. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*, September 10, 2023, New York, NY, USA. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3603269.3610855>

1 INTRODUCTION

Machine Learning (ML) has lately become a prominent research area for the networking community, with applications in a broad range of topics such as traffic classification [31], routing [8], congestion control [1], and traffic forecasting [19]. In particular, in-network ML has become very attractive, as it allows to leverage the expressiveness of ML models at data plane speed [28, 33]. The common denominator between many in-network ML use-cases is to train a model in the control plane using annotated historical data, and then deploy the model in the data plane for near real-time inference [21, 26, 33]. Unfortunately, the training data will eventually become outdated (a phenomenon formally known as “*distribution shift*” or “*concept drift*”), causing the deployed ML model to suffer from performance degradation [16, 17]. While the necessity for frequent model updates has already been raised [3], three fundamental

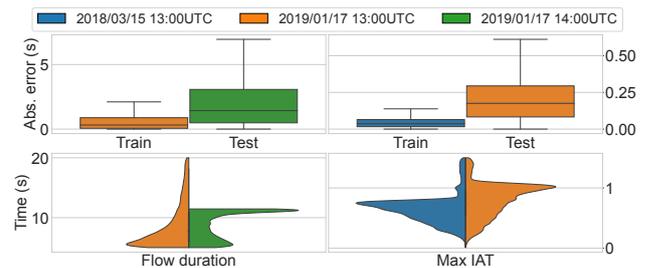


Figure 1: Distribution shift of TCP flow features from a real-world commercial backbone link [7]. The distribution shift of flow duration and inter-arrival time (IAT) causes performance degradation of a ML model trying to predict them.

questions remain: (1) **When should we update our model?** The answer is (in theory) fairly simple: *continuously*. We should assume that the input patterns observed by a deployed ML model may change at any point in time; (2) **Which data should we select for model updates?** Again, the answer is (in theory) simple: *only the data that is useful for learning new things*. (3) **What should our model learn?** In principle, *everything*. We want a model that dynamically expands its predictive power without forgetting past experiences.

In this poster, we aim to take a step towards designing a solution that answers those questions. We propose combining Active Learning (AL) [23], which enables filtering relevant information from a vast pool of unannotated data, and Continual Learning (CL) [10], which allows us to learn from streaming data *without forgetting past concepts*. The former, implemented in the switch ASIC, allows us to choose *the right amount of information* that shall be mirrored to the control plane, where the model is updated continually. Finally, the new model can be installed back in the data plane.

Implementing this solution is nontrivial and needs answering the following research questions: (1) how to implement AL-based filtering in the data plane?; (2) how selective should AL be for network learning?; (3) which ML models are most suitable for continual learning of network traffic?; and (4) how to dynamically reconfigure the data plane?

2 THE CASE FOR CONTINUAL LEARNING

We run some tests on real-world traces to characterize the amount of distribution shift in TCP flow features. We extracted commonly-used features from real-world traces [7] (e.g., flow duration, inter-arrival time (IAT), and packet size statistics). We observed a shift in the flow duration and in the maximum IAT for small time scales (~one hour) and for large time scales (~a year), respectively. To

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ACM SIGCOMM '23, September 10, 2023, New York, NY, USA

© 2023 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0236-5/23/09.

<https://doi.org/10.1145/3603269.3610855>

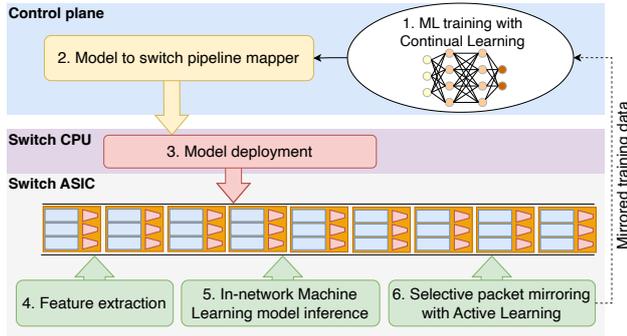


Figure 2: Our approach. The ML model is created (1) and then deployed in the switch ASIC (2), to perform inference at data plane speed (3). Selective mirroring (4) with Active Learning is deployed to keep the ML updated with Continual Learning.

quantify the impact of these shifts, we consider ad-hoc regression tasks¹ where the targets are either the flow duration or the maximum IAT. We observe² that the test error is significantly larger than training, a phenomenon that is imputable to the observed feature drifts (Fig. 1). Indeed, classical ML models will work properly only if the train and test data are approximately i.i.d. [5]. As such, practical in-network ML calls for smart, adaptive approaches.

Why can't we run existing proposals in a loop? Literature has been active in proposing efficient means for offloading trained ML models to the data plane [9, 28, 29, 34]. We here consider an orthogonal problem: how to train a ML model continually from packet streams with the optimal amount of annotated training data. Though Online Learning approaches have been explored [17, 28], they 1) assume that every streamed data point is labeled, and 2) do not pay attention about forgetting the past as long as the model is fit to the current experience. In our proposal, we want not only to learn adaptively, but also to remember (and therefore exploit) everything that was observed in the past. In this way, our model will not need additional data for re-learning already-observed concepts.

3 OUR APPROACH

Fig. 2 illustrates our proposal: to incorporate in a single closed-loop framework the following building blocks:

- (1) Model training: update the ML model over the time with CL.
- (2) Model deployment: deploy the new ML model in the data plane.
- (3) In-network inference: enable inference at data plane speed.
- (4) Selective mirroring: mirror to the control plane only the data useful for expanding the knowledge of the model with AL.

As a proof-of-concept experiment, we consider a subset of the CIC2019 dataset for DDoS classification [24]. We consider DDoS classes to represent disjoint learning tasks, which are presented to the model in sequence. For each task, the model must not only discriminate between benign and malicious flows but also place the malicious flows in the right class.

We implement a baseline Continual Random Forest (CRF), consisting of a RF augmented with a replay buffer storing the most informative past exemplars. We use the vote count as AL query

¹This is because CAIDA traces do not have task-specific class labels.

²For visualization purposes, we focus on the ranges [5, 20]s for flow duration and [0, 1.5]s for inter-arrival time.

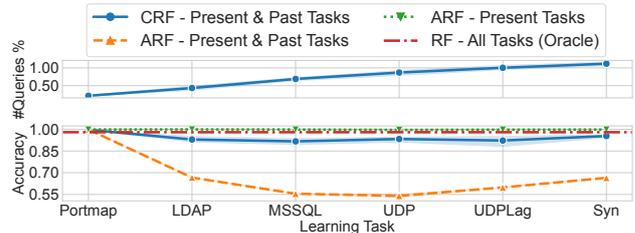


Figure 3: Adaptive vs. Continual Random Forests for class-incremental DDoS classification on CIC2019. At the end of the stream, CRF achieves performance close to an “oracle” while requiring only ~1% of the data.

strategy, selecting only data points whose predictions had less than 90% majority. We retrain after each query, which is computationally efficient for RFs. We consider an Adaptive Random Forest (ARF) as a purely online (but not continual) state-of-the-art baseline [11, 17]. In contrast to CRF, ARF assumes that every data point is labeled. We also consider an “oracle” RF trained on the full dataset as an upper-bound on the average performance over all tasks.

Fig. 3 shows the performance of CRF and ARF over the sequential tasks, and the percentage of queried labels by CRF relative to the full stream size. A purely adaptive learner such as ARF, though able to master individual tasks, quickly forgets past concepts. Instead, our baseline CRF achieves a performance close to the oracle upper-bound, while requiring labeling only ~1% of the observed samples.

4 CHALLENGES

Challenge #1: implementing AL-based filtering in the data plane. Vote count in our baseline CRF is a decent query strategy, but information-theoretic quantities [4, 12] are among the state-of-the-art. Their data plane implementation is not trivial, as it would require floating-point arithmetics. Even if not standard, authors in [18] propose a way to implement floating-point arithmetics in P4. **Challenge #2: how selective should AL be.** A small selectivity implies a large mirroring overhead, whereas a large selectivity implies a potential information loss. Applying AL to streaming data is, as of today, a novel twist on classical techniques [22]: investigating these trade-offs opens up interesting research directions.

Challenge #3: choosing the right CL strategy. Our baseline leverages a slowly-growing experience buffer, which may not be desirable. Strategies for maintaining the buffer of fixed size can be investigated [20]. Other solutions, e.g., regularized neural networks, do not require any storage overhead other than the model [15], but are ill-advised for tabular data [25]. Ultimately, the choice depends on the available storage/computational resources and the goodness-of-fit to the characteristics of task-specific data.

Challenge #4: runtime dataplane reconfiguration. Currently, if we want to add a new functionality to a switch, we need first to reroute the traffic of that switch, flush a new image in its ASIC and then restore the original traffic policy configuration. This process can lead to dramatic consequences if performed carelessly [14]. Programming the switch at run-time is possible [32], but not for RMT [6], the common commercial devices architecture [2, 13]. Researchers have also explored means to enable isolation between offloaded programs [27, 30, 35], which we will investigate to isolate the Active Learning processing and the rest of the pipeline.

REFERENCES

- [1] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. 2020. Classic Meets Modern: A Pragmatic Learning-Based Congestion Control for the Internet. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (Virtual Event, USA) (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 632–647. <https://doi.org/10.1145/3387514.3405892>
- [2] AMD. 2019. Naples DSC-100 Distributed Services Card. https://pensando.io/assets/documents/Naples_100_ProductBrief-10-2019.pdf
- [3] G. Apruzzese, P. Laskov, and J. Schneider. 2023. SoK: Pragmatic Assessment of Machine Learning for Network Intrusion Detection. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE Computer Society, Los Alamitos, CA, USA, 592–614. <https://doi.org/10.1109/EuroSP57164.2023.00042>
- [4] Freddie Bickford Smith, Andreas Kirsch, Sebastian Farquhar, Yarin Gal, Adam Foster, and Tom Rainforth. 2023. Prediction-Oriented Bayesian Active Learning. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 206)*, Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent (Eds.). PMLR, Valencia, Spain, 7331–7348. <https://proceedings.mlr.press/v206/bickfordsmith23a.html>
- [5] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.
- [6] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug 2013), 99–110. <https://doi.org/10.1145/2534169.2486011>
- [7] CAIDA. 2019. The CAIDA UCSD Anonymized Internet Traces - 2018-2019. https://www.caida.org/catalog/datasets/passive_dataset
- [8] Brian Chang, Aditya Akella, Loris D'Antoni, and Kausik Subramanian. 2023. Learned Load Balancing. In *Proceedings of the 24th International Conference on Distributed Computing and Networking (Kharagpur, India) (ICDCN '23)*. Association for Computing Machinery, New York, NY, USA, 177–187. <https://doi.org/10.1145/3571306.3571403>
- [9] Bruno Coelho and Alberto Schaeffer-Filho. 2022. BACKORDERS: Using Random Forests to Detect DDoS Attacks in Programmable Data Planes. In *Proceedings of the 5th International Workshop on P4 in Europe (Rome, Italy) (EuroP4 '22)*. Association for Computing Machinery, New York, NY, USA, 1–7. <https://doi.org/10.1145/3565475.3569074>
- [10] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. 2022. A Continual Learning Survey: Defying Forgetting in Classification Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 7 (2022), 3366–3385. <https://doi.org/10.1109/TPAMI.2021.3057446>
- [11] Heitor M. Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabricio Enembreck, Bernhard Pfahringer, and Talel Abdesslem. 2017. Adaptive random forests for evolving data stream classification. *Machine Learning* 106, 9 (Oct 2017), 1469–1495. <https://doi.org/10.1007/s10994-017-5642-8>
- [12] Neil Houlsby, Ferenc Huszar, Zoubin Ghahramani, and Máté Lengyel. 2011. Bayesian Active Learning for Classification and Preference Learning. arXiv:1112.5745 [stat.ML]
- [13] Intel. 2020. Tofino: P4-programmable Ethernet switch ASIC. <https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html>
- [14] Santosh Janardhan. 2021. Facebook Outage on October 4th 2021. <https://engineering.fb.com/2021/10/04/networking-traffic/outage/>
- [15] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114, 13 (2017), 3521–3526. <https://doi.org/10.1073/pnas.1611835114> arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1611835114
- [16] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. 2022. Matchmaker: Data Drift Mitigation in Machine Learning for Large-Scale Systems. In *Proceedings of Machine Learning and Systems*, D. Marculescu, Y. Chi, and C. Wu (Eds.), Vol. 4, 77–94. https://proceedings.mlsys.org/paper_files/paper/2022/file/069a002768bcb31509d4901961f23b3c-Paper.pdf
- [17] Pavol Mulinka and Pedro Casas. 2018. Adaptive Network Security through Stream Machine Learning. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos (Budapest, Hungary) (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 4–5. <https://doi.org/10.1145/3234200.3234246>
- [18] Shivam Patel, Rigden Atsatsang, Kenneth M. Tichauer, Michael H. L. S. Wang, James B. Kowalkowski, and Nik Sultana. 2022. In-Network Fractional Calculations Using P4 for Scientific Computing Workloads. In *Proceedings of the 5th International Workshop on P4 in Europe (Rome, Italy) (EuroP4 '22)*. Association for Computing Machinery, New York, NY, USA, 33–38. <https://doi.org/10.1145/3565475.3569083>
- [19] Yu Qiao, Chengxiang Li, Shuzheng Hao, Jun Wu, and Liang Zhang. 2022. Deep or Statistical: An Empirical Study of Traffic Predictions on Multiple Time Scales. In *Proceedings of the SIGCOMM '22 Poster and Demo Sessions (Amsterdam, Netherlands) (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 10–12. <https://doi.org/10.1145/3546037.3546048>
- [20] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. 2017. iCaRL: Incremental Classifier and Representation Learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 5533–5542. <https://doi.org/10.1109/CVPR.2017.587>
- [21] Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. 2018. Can the Network Be the AI Accelerator?. In *Proceedings of the 2018 Morning Workshop on In-Network Computing (Budapest, Hungary) (NetCompute '18)*. Association for Computing Machinery, New York, NY, USA, 20–25. <https://doi.org/10.1145/3229591.3229594>
- [22] Akanksha Saran, Safoora Yousefi, Akshay Krishnamurthy, John Langford, and Jordan T. Ash. 2023. Streaming Active Learning with Deep Neural Networks. arXiv:2303.02535 [cs.LG]
- [23] Burr Settles. 2012. *Active Learning*. Springer Cham, Cham, Switzerland. <https://doi.org/10.1007/978-3-031-01560-1>
- [24] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A. Ghorbani. 2019. Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCT)*. IEEE, Chennai, India, 1–8. <https://doi.org/10.1109/ICCT.2019.8888419>
- [25] Ravid Shwartz-Ziv and Amitai Armon. 2021. Tabular Data: Deep Learning is Not All You Need. In *8th ICML Workshop on Automated Machine Learning (AutoML)*. <https://openreview.net/forum?id=vdgtStp1Pv>
- [26] Giuseppe Siracusano and Roberto Bifulco. 2018. In-network Neural Networks. arXiv:1801.05731 [cs.DC]
- [27] Radostin Stoyanov and Noa Zilberman. 2020. MTPSA: Multi-Tenant Programmable Switches. In *Proceedings of the 3rd P4 Workshop in Europe (Barcelona, Spain) (EuroP4 '20)*. Association for Computing Machinery, New York, NY, USA, 43–48. <https://doi.org/10.1145/3426744.3431329>
- [28] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. 2022. Taurus: A Data Plane Architecture for per-Packet ML. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (Lausanne, Switzerland) (ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 1099–1114. <https://doi.org/10.1145/3503222.3507726>
- [29] Tushar Swamy, Annus Zulfiqar, Luigi Nardi, Muhammad Shahbaz, and Kunle Olukotun. 2023. Homunculus: Auto-Generating Efficient Data-Plane ML Pipelines for Datacenter Networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (Vancouver, BC, Canada) (ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 329–342. <https://doi.org/10.1145/3582016.3582022>
- [30] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. 2022. Isolation Mechanisms for High-Speed Packet-Processing Pipelines. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1289–1305. <https://www.usenix.org/conference/nsdi22/presentation/wang-tao>
- [31] Matthias Wichtlhuber, Eric Strehle, Daniel Kopp, Lars Prepens, Stefan Stegmüller, Alina Rubina, Christoph Dietzel, and Oliver Hohlfeld. 2022. IXP Scrubber: Learning from Blackholing Traffic for ML-Driven DDoS Detection at Scale. In *Proceedings of the ACM SIGCOMM 2022 Conference (Amsterdam, Netherlands) (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 707–722. <https://doi.org/10.1145/3544216.3544268>
- [32] Jiarong Xing, Kuo-Feng Hsu, Matty Kadosh, Alan Lo, Yonatan Piasetzky, Arvind Krishnamurthy, and Ang Chen. 2022. Runtime Programmable Switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 651–665. <https://www.usenix.org/conference/nsdi22/presentation/xing>
- [33] Zhaoyi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (Princeton, NJ, USA) (HotNets '19)*. Association for Computing Machinery, New York, NY, USA, 25–33. <https://doi.org/10.1145/3365609.3365864>
- [34] Changgang Zheng, Zhaoyi Xiong, Thanh T Bui, Siim Kaupmees, Riyad Bensousane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. 2022. IIsy: Practical In-Network Classification. arXiv:2205.08243 [cs.NI]
- [35] Peng Zheng, Theophilus Benson, and Chengchen Hu. 2018. P4Visor: Lightweight Virtualization and Composition Primitives for Building and Testing Modular Programs. In *Proceedings of the 14th International Conference on Emerging Networking Experiments and Technologies (Heraklion, Greece) (CoNEXT '18)*. Association for Computing Machinery, New York, NY, USA, 98–111. <https://doi.org/10.1145/3281411.3281436>

7 | Large Language Models for Automated Network Configuration

The advent of Large Language Models has revolutionized the landscape of AI by enabling opportunities for automation unachievable by prior technologies. Networking is no exception, and early research is investigating the potential of LLM for automating specific network management tasks, e.g., alarm log analysis [185] and network configuration [184]. This paper considers the task of network configuration, specifically focusing on the problem of how to cheaply and efficiently teach open-source models (i.e., LLM whose weights are public) how to configure an optical network testbed. To solve this problem efficiently, we decompose the task of generating a complete network configuration to a planning step, i.e., decompose the full task into a list of atomic interactions with the testbed (e.g., set up a lightpath), followed by an execution step, i.e., for each atomic task, generate the data structure that realizes that task in the testbed controller. This allows the LLM to reason over small-sized tasks, thus reducing the surface of errors. Moreover, we observed empirically that the majority of the LLM's errors were syntactical, i.e., the model would struggle to respect user-provided data formats (e.g., a JSON schema). To solve this problem, we constrained the generation process of the LLM at runtime such that the produced data structures were guaranteed to respect JSON schema specifications. Overall, our proposed system was able to achieve over 80% accuracy using state-of-the-art open-source models, and this performance metric is likely to improve as models become more capable over time. The dataset and the methodologies developed in this work played a critical role in developing a follow-up post-deadline paper [51].

Open Implementation of a Large Language Model Pipeline for Automated Configuration of Software-Defined Optical Networks

Nicola Di Cicco*, Memedhe Ibrahim, Sebastian Troia, Francesco Musumeci, Massimo Tornatore

Politecnico di Milano, Milan, Italy, * corresponding author: nicola.dicicco@polimi.it

Abstract We leverage LLMs to develop a natural-language interface to a software-defined optical network testbed. Results show over 80% accuracy in translating human intent to the appropriate network configurations. Our code is public. ©2024 The Author(s)

Introduction

Large Language Models (LLMs) have experienced groundbreaking progress in recent years, achieving state-of-the-art performance in many natural language and code generation tasks^[1]. In particular, tremendous advances in open-source LLMs make it possible to execute LLMs *locally* on commodity hardware while rivaling the performance of state-of-the-art proprietary solutions such as GPT-3.5^{[2],[3]}. As such, LLMs hold vast untapped potential for developing innovative LLM-powered applications, with software-defined optical networking standing as no exception.

In this paper, for the first time to the best of our knowledge, we build and make public an LLM-based application realizing a natural-language interface to a software-defined optical-network (SDON) testbed. The purpose of the application is to translate open-ended natural language queries from a user (e.g., “set up a lightpath”, or “measure the OSNR of a service”) into domain-specific data models to be sent through an SDON, and to provide feedback on the outcome of the user’s request. While our application targets specifically operations on our testbed, the methodology and the practical guidelines distilled in this paper are generalizable to any software-defined testbed exposing a well-defined Northbound Interface (NBI). We make our code (entirely built on open-source software and open-source LLMs), data, and scripts to reproduce experiments publicly available to foster future research in this field.¹

Building an LLM-based interface for automated network configuration demands a specialized knowledge of the inner workings of both LLMs and the optical-network testbed. Here, we identify two fundamental challenges and anticipate our proposed working solutions to solve them.

Challenge 1: the LLM has zero prior knowledge about controlling any similar testbed. This is a sensible assumption, since domain-specific documentation of optical-network testbeds is, in general, not publicly available on the Internet, and hence cannot be present in the text corpora

used to train the LLM. We therefore face the challenge of how to efficiently inject new knowledge into pre-trained LLMs.

Challenge 2: LLM outputs are not controllable (i.e., “hallucinations” occur). Regrettably, LLMs are prone to producing non-factual outputs. Though “prompt engineering” techniques (i.e., designing task-specific inputs) can mitigate this phenomenon, we have no guarantees that LLMs will never hallucinate. When interacting with a testbed, hallucinations can translate into malformed data in the best case, and catastrophically destructive actions in the worst case. For this reason, we need to implement fail-safe mechanisms to avoid undesired outcomes.

Teaching new skills to pre-trained LLMs

A first strategy for incorporating new knowledge into an LLM is through **fine-tuning** via Supervised Learning on a target training dataset. This has two main limitations: 1) it requires costly GPUs (though this need can be partially circumvented with modern approximating algorithms^{[4],[5]}), and 2), most importantly, it requires sizeable datasets high-quality prompt-answer exemplars^{[4],[5]}, which may be extremely time-consuming to collect, e.g., in the case of instructions for SDON configuration.

Hence, we decided to adopt a second strategy, called **in-context learning**, i.e., all the information relevant to solving the new task is provided in the LLM’s prompt. The intuition is that LLMs, being trained on colossal text corpora, become capable of *learning new skills from analogy* at runtime. One can think of prompting as a way of “programming” LLMs. Moreover, even though in-context learning is limited by the LLM’s *context window* (i.e., its short-term memory), recent breakthroughs achieved context windows as long as 700.000 words^[6]. However, even though in-context learning is an exceptionally powerful and versatile technique, naive prompting (e.g., “do X; context: Y”) will result in sub-optimal outputs, mainly due to hallucinations^[7]. In the following, we outline our proposed solutions to create valid outputs and minimize hallucinations.

¹ <https://github.com/nicoladicicco/llm-orchestrator>

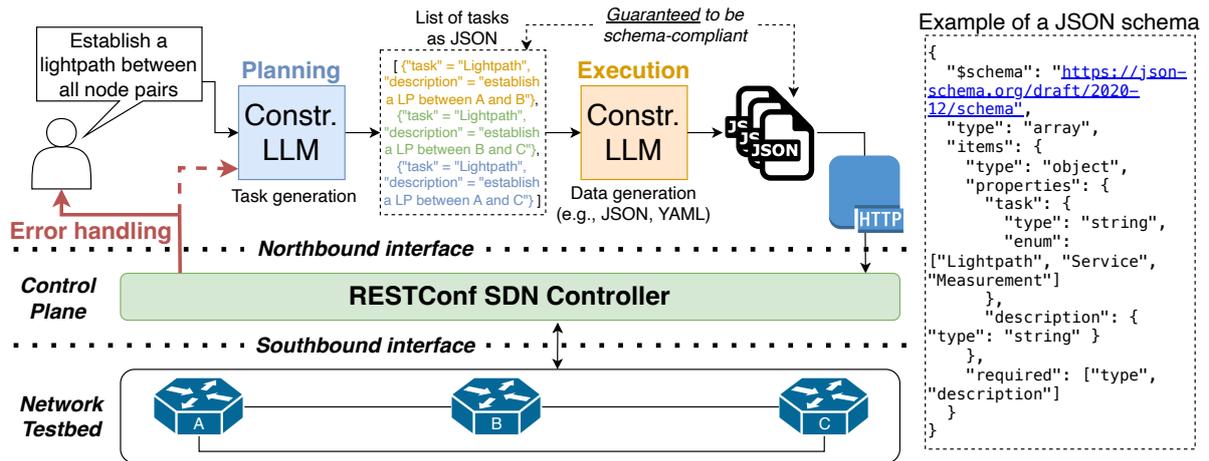


Fig. 1: System design of an LLM-based interface to a software-defined optical network testbed.

Designing an LLM-based interface for software-defined optical networks

Our core idea is as follows: while we cannot guarantee that the LLM’s output will exactly realize the user’s intent, we can guarantee that it will always be *valid*, i.e., well-formed, according to a user-specified data model. To achieve this goal, we leverage compositional learning^{[8]–[11]}, i.e., we decompose “translating human intent in an appropriate data structure” in three simpler tasks: 1) **Planning**, where an LLM translates human intent into a list of tasks, 2) **Execution**, where an LLM produces the appropriate data structure for each task, and 3) **Error handling**, where an LLM attempts to rectify easily-fixable errors with minimal human supervision. All phases leverage **Constrained Generation**, which ensures that the produced outputs are valid. Fig. 1 illustrates our proposed system design.

Step 1: Planning. We transform the user’s query into a list of atomic tasks, which we represent as JSON data. We design a JSON schema with two fields: “task”, which takes values in a finite set of keywords (in our case, “Lightpath”, “Service” or “Measurement”), and “description”, which is a summary in natural language of the task and of its requirements. Intuitively, we can think of the “prompt → task” mapping as sentence classification, and the “prompt → description” mapping as summarization/paraphrasing, both tasks at which LLMs naturally excel.

Step 2: Execution. We loop over each task generated in Planning and perform the following operations: (i) from the “task” field, we retrieve the corresponding data model (e.g., if “task” is “Lightpath”, we retrieve a JSON schema defining the data model for instantiating a lightpath); (ii) we provide the “description” field alongside the retrieved data model to the LLM. The output is a data structure (e.g., a JSON list) realizing the task’s requirements, which can then be encapsulated in a REST message and sent through the NBI.

Step 3: Error handling. Even though the messages generated by the LLM are syntactically correct, the SDN controller may not succeed in handling the generated requests (e.g., lightpath creation fails), returning an error. One option could be to return control to the user for manual intervention. Instead, we propose to let the LLM attempt to fix (some of) the errors. For instance, simple errors such as “lightpath ID 42 is already taken” can be easily fixed by choosing a different ID. As a simple solution, we can concatenate the error body to the input prompt, turning the error message into additional requirements, and restart generation from the planning phase. Note that most classes of errors (e.g., “no spectrum available between nodes A and B”) cannot be trivially fixed without human guidance. In these cases, one can instruct the LLM to return control in case of specific error types, or (more challenging) let the LLM decide to return control if the input context is not sufficient to resolve the error. Our solution covers only a limited set of errors, leaving ample room for future research (e.g., in repeated user-LLM interaction.)

Constrained generation. We leverage *formal grammars* to constrain at runtime the generation process, such that each output is *guaranteed* to be a valid data structure under a user-specified data model. In particular, we leverage JSON Schemas, which we convert to GGML Bakus-Naur Form (GBNF) grammars^[3]. During generation, we use the grammar to dynamically restrict the LLM’s output dictionary according to the constraints specified by the JSON schema and the current generated text. As an example, consider the JSON schema in Fig. 1. When generating the field “task”, we restrict the LLM’s output to choose only among the specified admissible words, i.e., “Lightpath”, “Service”, or “Measurement”. This grammar-based approach can be generalized to arbitrary domain-specific constraints (e.g., generating syntactically valid YAML files), providing an effective way for enforcing “guardrails” to the LLM’s output.

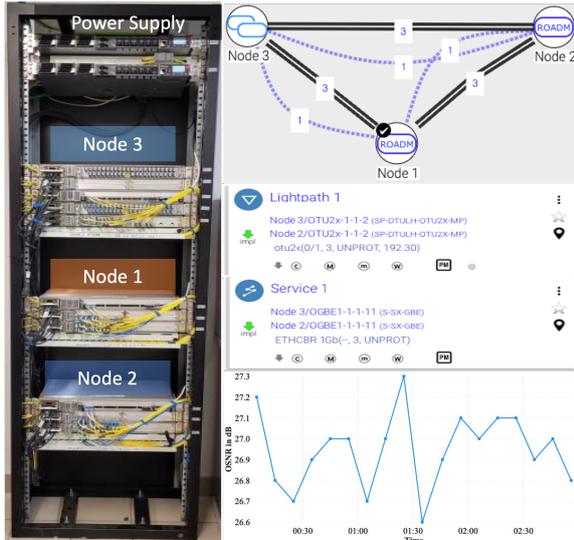


Fig. 2: Filterless optical network testbed. Solid and dashed lines represent fiber connections and lightpaths, respectively.

Numerical Results

Testbed description. Our testbed is a three-node optical filterless ring network equipped with 10G (non-coherent) transponders. The testbed comprises WDM-layer equipment (to establish lightpaths between source-destination pairs) and OTN-layer equipment (to provision services that carry traffic at higher protocol layers), managed by the SDON controller and an orchestrator developed within our laboratory. Northbound and southbound interfaces are implemented using REST-Conf. We consider three functionalities of the SDON controller, i.e., 1) lightpath establishment, 2) service provisioning, and 3) performance monitoring. Lightpaths can be established between any node pair through 10G transponders, while services carrying traffic may be 1G or 10G. Measurement campaigns can be initiated on established lightpaths and provisioned services, with 15-minute or 24-hour granularities. Fig. 2 illustrates our testbed. We show a successful implementation of *Lightpath 1* and *Service 1* between Node 2 and Node 3, and the OSNR measured at the receiver for *Lightpath 1*.

Performance evaluation. We use Mixtral-8x7B^[2] as an LLM model. We manually curate a dataset including 50 prompt-answer pairs. The answers consist of JSON files realizing the question’s requirements. Each question is marked as Easy, Medium, or Hard based on the length of the correct answer, as longer answers imply more tasks and/or requirements. We consider a generated answer to be “correct” if all the requirements are satisfied and “incorrect” if some requirements are not satisfied and/or the model hallucinates content beyond the user’s request. Specifically, we consider four types of incorrect answers, i.e., “Invalid JSON”, “Missing Tasks”, “Surplus Tasks”, and “Missing Requirements”. We compare against

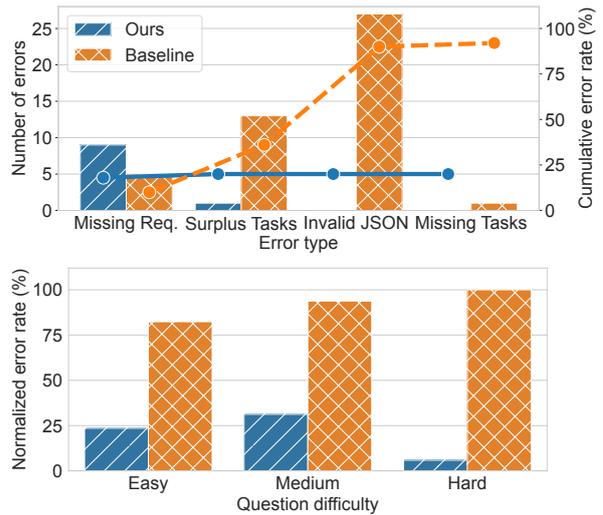


Fig. 3: Performance evaluation of our LLM-based pipeline for automated optical-network configuration.

a baseline leveraging in-context learning without task decomposition and constrained generation.

Fig. 3 shows our performance evaluation results. The baseline achieves a staggering 92% error rate, showing that LLMs are not plug-and-play tools for automated network configuration. In contrast, our approach achieves a 20% error rate. Moreover, while the baseline’s performance gets worse with the answer’s difficulty, our approach does not show a distinct trend. Indeed, the baseline has to generate long data structures attempting to respect the user’s requirements and the JSON schema, while our approach generates a series of self-contained, small data structures. We then manually inspected the incorrect responses produced by our system. In eight out of nine “Missing Requirements” errors, the user’s request could have been interpreted ambiguously. For instance, if the user asks “the ConfigurationState should be defined”, it can either mean “the field ConfigurationState should be present in the JSON”, or “set ‘defined’ as the value of the ConfigurationState field”, with the latter being the intended interpretation. Minor paraphrasing solves these errors, reducing the total error rate to 4%. This leaves out two error instances: one where the model creates two services with the same name, and one where the model creates an unsolicited measurement campaign. The first error can be easily detected by the control plane and communicated with a message such as “Error: cannot create two services with the same name.” By adding this message to the input prompt in the planning phase, the LLM generates a correct answer, achieving a 2% error rate. To conclude, another advantage of our task decomposition approach is inference time. On a high-end laptop, the baseline requires 90s to produce an output due to having to process a long input prompt. Instead, our approach requires less than 30s (a 3x speedup), with the main bottleneck being text generation.

Acknowledgements

This work was partly supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on “Telecommunications of the Future” (PE00000001 - program “RESTART”) and by the PRIN project ZeTON, funded by Italian Ministry of University and Research.

References

- [1] OpenAI *et al.*, *GPT-4 technical report*, 2024. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs.CL].
- [2] A. Q. Jiang *et al.*, *Mixtral of experts*, 2024. DOI: 10.48550/arXiv.2401.04088. arXiv: 2401.04088 [cs.LG].
- [3] G. Gerganov, *Llama.cpp*, <https://github.com/ggerganov/llama.cpp/>, 2024.
- [4] E. J. Hu, Y. Shen, P. Wallis, *et al.*, “Lora: Low-rank adaptation of large language models”, in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022.
- [5] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “Qlora: Efficient finetuning of quantized llms”, in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., 2023, pp. 10 088–10 115.
- [6] Google, *Gemini 1.5*, <https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/>, 2024.
- [7] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, “Why johnny can’t prompt: How non-ai experts try (and fail) to design llm prompts”, in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023. DOI: 10.1145/3544548.3581388.
- [8] J. Loula, M. Baroni, and B. Lake, “Rearranging the familiar: Testing compositional generalization in recurrent networks”, in *EMNLP Workshop on BlackboxNLP*, 2018.
- [9] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, “HuggingGPT: Solving AI tasks with chatGPT and its friends in hugging face”, in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
- [10] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, “Llm-planner: Few-shot grounded planning for embodied agents with large language models”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2023.
- [11] P. Lu, B. Peng, H. Cheng, *et al.*, “Chameleon: Plug-and-play compositional reasoning with large language models”, in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., 2023, pp. 43 447–43 478.

8 | Conclusion and future developments

8.1. Conclusion

In this Thesis, we introduced five fundamental open challenges in ML applied to network management: generalizability, adaptability, data efficiency, reliability, and performance. Considering multiple representative use cases in network management, ranging from combinatorial optimization to traffic classification, we identified the most pertinent challenges and proposed novel technical solutions. We now briefly review our main technical contributions and summarize our lessons learned.

ML for Network Optimization. To be practically useful, ML-based solving algorithms must be generalizable to problem instances different (and possibly larger) than training and perform favorably against battle-tested approaches. We numerically assessed the limitations of end-to-end ML in addressing these challenges. Instead, we found that the integration of ML with existing optimization strategies such as Local Search, together with a model architecture able to handle variable-sized inputs and outputs, consistently outperformed state-of-the-art metaheuristics. We conclude that integrating ML with OR emerges as the most promising direction for developing data-driven network optimization algorithms. We recommend future research to investigate other flavors of integrating ML with classical optimization strategies, such as Genetic Algorithms or Variable Neighbourhood Search algorithms.

ML for Failure Identification. We highlighted that the main challenges in applying ML to network fault identification are not fitting models, but collecting sufficient good-quality training data and ensuring that the model provides reliable predictions once deployed. We addressed data efficiency by developing data augmentation and Active Learning algorithms, and reliability by using Venn-Abers predictors on top of existing trained models. Our contributions resulted in significantly lower model training costs, and in models with mathematically-guaranteed reliability properties, which we extensively evaluated on real-

world datasets. Generally speaking, focusing on what happens before and after model fitting opens up many interesting research opportunities. For instance, problems such as Online Learning or model explainability are only partially addressed in the literature of failure identification.

ML for Service Orchestration. We considered the application of ML to online resource allocation problems under dynamic traffic. Similarly to offline optimization, we found it critical for generalizability to use model architectures that can handle inputs and outputs of variable size. Moreover, we framed online resource allocation as a multi-objective problem, and proposed a ML-based algorithm for automatically exploring the space of possible trade-offs. Numerical results illustrate that our methodologies outperform standard heuristics and generalize to networks much larger than training. We recommend further investigating ML-based algorithms for online multi-objective resource allocation, e.g., jointly accounting for cost, reliability, and QoS, which are of great practical interest.

Continual In-Network ML. We considered the problem of data drift in ML models deployed onto programmable switches. Specifically, we addressed data-efficient adaptation to streams of fresh data. To solve this challenge, we proposed implementing Online Active Learning in the data plane to dynamically filter the most informative data from a stream. Despite the hardware limitations of programmable switches, we show that a naive Online Active Learning algorithm can achieve continual adaptation with a fraction of the training data. More sophisticated in-network Active Learning strategies, and their implementation on real hardware, are presently yet to be explored.

LLMs for Network Automation. We investigated using pre-trained LLMs to autonomously configure a software-defined network based on natural-language instructions, e.g., "Set up a lightpath between nodes A and B." We leveraged in-context learning to inject domain-specific knowledge to the LLM without the need for retraining, and task decomposition to achieve a sufficiently high level of accuracy. In general, we found that LLMs excel when used as a means to call a set of external tools (e.g., via API calls). We expect future research to further investigate this topic, e.g., by using LLMs to jointly interface with configuration, optimization and monitoring software.

Though algorithms presented in this Thesis are tailored for our considered application scenarios, we expect our distilled insights and design principles to generalize well across different application domains. Most of our code and data, including real-world datasets curated by industry partners, has been made public:¹ we hope that the research community, as well as the industry, will make liberal use of our work.

¹<https://github.com/bonsai-lab-polimi>

8.2. Future developments

I am currently playing a central role in the definition of two large-scale projects in collaboration with industrial partners. These projects extend the methodologies proposed in this Thesis regarding In-Network Continual Learning and LLM-based network automation. A brief description of the projects is as follows:

Decentralized ML-based control of RDMA networks for LLM training: as the parameter count of future LLMs is prospected to increase, so is the scale of networked datacenter systems required for training (recently, X/Twitter announced the deployment of a cluster comprising 100.000 NVIDIA H100 GPUs). In this context, networking, especially the interconnection between datacenters on heterogeneous physical premises, becomes a major bottleneck for LLM training, resulting in hardware underutilization. This project will investigate ML-based methods for optimally scheduling LLM training data transfers through intra- and inter-DC networks. The ML-based engine will be designed to be 1) decentralized, i.e., able to work without the presence of a centralized controller / scheduler; 2) continually adaptive, i.e., able to dynamically update its decision rules based on the current observed network state; 3) natively compatible with state-of-the-art networking technologies for hardware-accelerated LLM training, e.g., RDMA [189–192]. This project has been awarded funding and the work is presently ongoing.

LLMs for zero-touch software-defined optical networking. We will extend our contributions presented at the ECOC 2024 conference [45, 51] with experimental deployment on large-scale optical-network testbeds, and by increasing the portfolio of use-cases supported by our LLM-based system. Specifically, we aim at considering 1) automated network troubleshooting, leveraging LLM for automated log analysis and countermeasure proposal, and 2) automated optical-network optimization, where the LLM will repeatedly interact with the network to achieve a user-specified desired optimized state (e.g., power and spectrum configuration achieving minimal interference). We plan to design ad-hoc datasets and novel performance metrics for quantitative performance evaluations.

Bibliography

- [1] Francesco Musumeci, Cristina Rottondi, Avishek Nag, Irene Macaluso, Darko Zibar, Marco Ruffini, and Massimo Tornatore. An overview on application of machine learning techniques in optical networks. *IEEE Communications Surveys & Tutorials*, 21(2):1383–1408, 2019.
- [2] Yaohua Sun, Mugen Peng, Yangcheng Zhou, Yuzhe Huang, and Shiwen Mao. Application of machine learning in wireless networks: Key techniques and open issues. *IEEE Communications Surveys & Tutorials*, 21(4):3072–3108, 2019.
- [3] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. DOTE: Rethinking (predictive) WAN traffic engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1557–1581, Boston, MA, April 2023. USENIX Association.
- [4] Zhiying Xu, Francis Y. Yan, Rachee Singh, Justin T. Chiu, Alexander M. Rush, and Minlan Yu. Teal: Learning-accelerated optimization of wan traffic engineering. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, pages 378–393, New York, NY, USA, 2023. Association for Computing Machinery.
- [5] Abd AlRhman AlQiam, Yuanjun Yao, Zhaodong Wang, Satyajeet Singh Ahuja, Ying Zhang, Sanjay G. Rao, Bruno Ribeiro, and Mohit Tawarmalani. Transferable neural wan te for changing topologies. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, pages 86–102, New York, NY, USA, 2024. Association for Computing Machinery.
- [6] Ximeng Liu, Shizhen Zhao, Yong Cui, and Xinbing Wang. Figret: Fine-grained robustness-enhanced traffic engineering. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, pages 117–135, New York, NY, USA, 2024. Association for Computing Machinery.
- [7] Zhaoqi Xiong and Noa Zilberman. Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM Workshop on Hot Topics*

- in Networks*, HotNets '19, page 25–33, New York, NY, USA, 2019. Association for Computing Machinery.
- [8] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. Taurus: A data plane architecture for per-packet ml. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 1099–1114, New York, NY, USA, 2022. Association for Computing Machinery.
- [9] Qizheng Zhang, Ali Imran, Enkeleda Bardhi, Tushar Swamy, Nathan Zhang, Muhammad Shahbaz, and Kunle Olukotun. Caravan: Practical online learning of In-Network ML models with labeling agents. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 325–345, Santa Clara, CA, July 2024. USENIX Association.
- [10] Syed Usman Jafri, Sanjay Rao, Vishal Shrivastav, and Mohit Tawarmalani. Leo: Online ML-based traffic classification at Multi-Terabit line rate. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1573–1591, Santa Clara, CA, April 2024. USENIX Association.
- [11] Davide Andreoletti, Sebastian Troia, Francesco Musumeci, Silvia Giordano, Guido Maier, and Massimo Tornatore. Network traffic prediction based on diffusion convolutional recurrent neural networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 246–251, 2019.
- [12] Rodolfo Alvizu, Sebastian Troia, Guido Maier, and Achille Pattavina. Matheuristic with machine-learning-based prediction for software-defined mobile metro-core networks. *Journal of Optical Communications and Networking*, 9(9):D19–D30, 2017.
- [13] Abdelhadi Azzouni and Guy Pujolle. Neutm: A neural network-based framework for traffic matrix prediction in sdn. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5, 2018.
- [14] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-Learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, Renton, WA, April 2018. USENIX Association.
- [15] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International*

- Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3050–3059. PMLR, 09–15 Jun 2019.
- [16] Alessio Sacco, Matteo Flocco, Flavio Esposito, and Guido Marchetto. Owl: Congestion control with partially invisible networks via reinforcement learning. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.
- [17] Chen-Yu Yen, Soheil Abbasloo, and H. Jonathan Chao. Computers can learn from the heuristic designs and master internet congestion control. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 255–274, New York, NY, USA, 2023. Association for Computing Machinery.
- [18] Shahin Shahkarami, Francesco Musumeci, Filippo Cugini, and Massimo Tornatore. Machine-learning-based soft-failure detection and identification in optical networks. In *2018 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3, 2018.
- [19] Francesco Musumeci, Cristina Rottondi, Giorgio Corani, Shahin Shahkarami, Filippo Cugini, and Massimo Tornatore. A tutorial on machine learning for failure management in optical networks. *Journal of Lightwave Technology*, 37(16):4125–4139, 2019.
- [20] Zhuotong Li, Yongli Zhao, Yajie Li, Sabidur Rahman, Xiaosong Yu, and Jie Zhang. Demonstration of fault localization in optical networks based on knowledge graph and graph neural network. In *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3, 2020.
- [21] 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*, 10(2):A286–A297, 2018.
- [22] Memedhe Ibrahim, Hatef Abdollahi, Cristina Rottondi, Alessandro Giusti, Alessio Ferrari, Vittorio Curri, and Massimo Tornatore. Machine learning regression for qot estimation of unestablished lightpaths. *Journal of Optical Communications and Networking*, 13(4):B92–B101, 2021.
- [23] Matteo Salani, Cristina Rottondi, Leopoldo Ceré, and Massimo Tornatore. Dual-stage planning for elastic optical networks integrating machine-learning-assisted qot estimation. *IEEE/ACM Transactions on Networking*, 31(3):1293–1307, 2023.

- [24] Jingjing Wang, Chunxiao Jiang, Haijun Zhang, Yong Ren, Kwang-Cheng Chen, and Lajos Hanzo. Thirty years of machine learning: The road to pareto-optimal wireless networks. *IEEE Communications Surveys & Tutorials*, 22(3):1472–1514, 2020.
- [25] Junfeng Xie, F. Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, and Yunjie Liu. A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges. *IEEE Communications Surveys & Tutorials*, 21(1):393–430, 2019.
- [26] Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN international conference on functional programming*, pages 1–1, 2016.
- [27] 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.
- [28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.
- [29] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [30] Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27):e2311878121, 2024.
- [31] Ulf Johansson, Tuwe Löfström, Håkan Sundell, Henrik Linusson, Anders Gidenstam, and Henrik Boström. Venn predictors for well-calibrated probability estimation trees. In *Conformal and Probabilistic Prediction and Applications*, pages 3–14. PMLR, 2018.
- [32] Yu Bai, Song Mei, Huan Wang, and Caiming Xiong. Don’t just blame over-parametrization for over-confidence: Theoretical analysis of calibration in binary classification. In *International conference on machine learning*, pages 566–576. PMLR, 2021.
- [33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with

- deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
- [34] Nicola Di Cicco, Emre Furkan Mercan, Oleg Karandin, Omran Ayoub, Sebastian Troia, Francesco Musumeci, and Massimo Tornatore. On deep reinforcement learning for static routing and wavelength assignment. *IEEE Journal of Selected Topics in Quantum Electronics*, 28(4):1–12, 2022.
- [35] Nicola Di Cicco, Memedhe Ibrahim, Sebastian Troia, and Massimo Tornatore. Deepls: Local search for network optimization based on lightweight deep reinforcement learning. *IEEE Transactions on Network and Service Management*, 21(1):108–119, 2024.
- [36] Nicola Di Cicco, Memedhe Ibrahim, Francesco Musumeci, Federica Bruschetta, Michele Milano, Claudio Passera, and Massimo Tornatore. Machine learning for failure management in microwave networks: A data-centric approach. *IEEE Transactions on Network and Service Management*, pages 1–1, 2024.
- [37] Nicola Di Cicco, Memedhe Ibrahim, Omran Ayoub, Federica Bruschetta, Michele Milano, Claudio Passera, and Francesco Musumeci. Asap hardware failure-cause identification in microwave networks using venn-abers predictors. *IEEE Transactions on Network and Service Management*, pages 1–1, 2024.
- [38] Amazon mechanical turk. <https://www.mturk.com/>. Accessed: 2024-10-03.
- [39] Vladimir Vovk, Ivan Petej, and Valentina Fedorova. Large-scale probabilistic predictors with and without guarantees of validity. *Advances in Neural Information Processing Systems*, 28, 2015.
- [40] Nicola Di Cicco, Gaetano Francesco Pittalà, Gianluca Davoli, Davide Borsatti, Walter Cerroni, Carla Raffaelli, and Massimo Tornatore. Drl-forch: A scalable deep reinforcement learning-based fog computing orchestrator. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pages 125–133, 2023.
- [41] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions. *IEEE Communications Surveys & Tutorials*, 23(4):2557–2589, 2021.
- [42] Gianluca Davoli, Davide Borsatti, Daniele Tarchi, and Walter Cerroni. Forch: An orchestrator for fog computing service deployment. In *2020 IFIP Networking Conference (Networking)*, pages 677–678, 2020.

- [43] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.
- [44] Nicola Di Cicco, Amir Al Sadi, Chiara Grasselli, Andrea Melis, Gianni Antichi, and Massimo Tornatore. Poster: Continual network learning. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 1096–1098, New York, NY, USA, 2023. Association for Computing Machinery.
- [45] Nicola Di Cicco, Memedhe Ibrahimi, Sebastian Troia, Francesco Musumeci, and Massimo Tornatore. Open implementation of a large language model pipeline for automated configuration of software-defined optical networks. In *50th European Conference on Optical Communications (ECOC)*, 2024.
- [46] Tom B Brown et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- [47] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [48] Alexander Clemm, Laurent Ciavaglia, L Zambenedetti Granville, and Jeff Tantsura. Rfc 9315: Intent-based networking-concepts and definitions, 2022.
- [49] Roy T Fielding and Richard N Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.
- [50] Memedhe Ibrahimi, Keerthikumaran Selvamuthukumaran, Sebastian Troia, Massimo Tornatore, and Francesco Musumeci. Automated data ingestion framework for enhanced control and maintenance of optical networks [invited]. In *2024 24th International Conference on Transparent Optical Networks (ICTON)*, pages 1–4, 2024.
- [51] Chenyu Sun, Xin Yang, Nicola Di Cicco, Reda Ayassi, Venkata Virajit Garbhapu, Photios A. Stavrou, Massimo Tornatore, Gabriel Charlet, and Yvan Pointurier. First demonstration of fine-tuned llm for digital twin optical networks: Ai-agent for lifecycle management and automation. In *50th European Conference on Optical Communications (ECOC)*, 2024.
- [52] Nicola Di Cicco, Mëmédhe Ibrahimi, Cristina Rottondi, and Massimo Tornatore. Calibrated probabilistic qot regression for unestablished lightpaths in optical net-

- works. In *2022 International Balkan Conference on Communications and Networking (BalkanCom)*, pages 21–25, 2022.
- [53] Nicola Di Cicco, Federico Tonini, Valentina Cacchiani, and Carla Raffaelli. Optimization over time of reliable 5g-ran with network function migrations. *Computer Networks*, 215:109216, 2022.
- [54] Omran Ayoub et al. Explainable artificial intelligence in communication networks: A use case for failure identification in microwave networks. *Computer Networks*, 219:109466, 2022.
- [55] Nicola Di Cicco, Simone Del Prete, Silvi Kodra, Marina Barbiroli, Franco Fuschini, Enrico M. Vitucci, Vittorio Degli Esposti, and Massimo Tornatore. Machine learning-based line-of-sight prediction in urban manhattan-like environments. In *2023 17th European Conference on Antennas and Propagation (EuCAP)*, pages 1–5, 2023.
- [56] Nicola Di Cicco, Jacopo Talpini, Mëmëdhe Ibrahimi, Marco Savi, and Massimo Tornatore. Uncertainty-aware qot forecasting in optical networks with bayesian recurrent neural networks. In *ICC 2023 - IEEE International Conference on Communications*, pages 441–446, 2023.
- [57] Giovanni Sticca, Memedhe Ibrahimi, Francesco Musumeci, Nicola Di Cicco, Andrea Castoldi, Rosanna Pastorelli, and Massimo Tornatore. Selective hybrid edfa/raman amplifier placement to mitigate lightpath degradation in $(c + 1)$ networks. *Journal of Optical Communications and Networking*, 15(8):C232–C241, 2023.
- [58] Faruk Pasic, Nicola Di Cicco, Marco Skocaj, Massimo Tornatore, Stefan Schwarz, Christoph F. Mecklenbräuker, and Vittorio Oegli-Esposti. Multi-band measurements for deep learning-based dynamic channel prediction and simulation. *IEEE Communications Magazine*, 61(9):98–104, 2023.
- [59] Marco Skocaj, Nicola Di Cicco, Tommaso Zugno, Mate Boban, Jiri Blumenstein, Ales Prokes, Tomas Mikulasek, Josef Vychodil, Konstantin Mikhaylov, Massimo Tornatore, and Vittorio Degli-Esposti. Vehicle-to-everything (v2x) datasets for machine learning-based predictive quality of service. *IEEE Communications Magazine*, 61(9):106–112, 2023.
- [60] Aryanaz Attarpour, Memedhe Ibrahimi, Nicola Di Cicco, Francesco Musumeci, Andrea Castoldi, Mario Ragni, and Massimo Tornatore. Minimizing the cost of hierarchical optical transport network traffic grooming boards in metro networks. *Journal of Optical Communications and Networking*, 15(10):E18–E28, 2023.

- [61] G. S. Sticca, M. Ibrahimi, N. Di Cicco, F. Musumeci, and M. Tornatore. Throughput maximization in (c+l+s) networks with incremental deployment of hfas and 3rs. In *49th European Conference on Optical Communications (ECOC 2023)*, volume 2023, pages 1465–1468, 2023.
- [62] José Santos, Mattia Zaccarini, Filippo Poltronieri, Mauro Tortonesi, Cesare Sleianelli, Nicola Di Cicco, and Filip De Turck. Efficient microservice deployment in kubernetes multi-clusters through reinforcement learning. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pages 1–9, 2024.
- [63] Giovanni Simone Sticca, Memedhe Ibrahimi, Nicola Di Cicco, Francesco Musumeci, and Massimo Tornatore. Hollow-core-fiber placement in latency-constrained metro networks with edgedcs. In *2024 Optical Fiber Communications Conference (OFC)*, pages 1–3, 2024.
- [64] Aryanaz Attarpour, Memedhe Ibrahimi, Nicola Di Cicco, Francesco Musumeci, Andrea Castoldi, Mario Ragni, and Massimo Tornatore. Joint qot-aware optimization of otn and wdm layers for low-cost optical metro networks. In *ICC 2024 - IEEE International Conference on Communications*, pages 4979–4984, 2024.
- [65] Klevis Duka, Nicola Di Cicco, and Enrico M. Vitucci. A study on the use of convolutional networks for rf coverage evaluations in urban environments. In *2024 IEEE International Symposium on Antennas and Propagation and INC/USNC-URSI Radio Science Meeting (AP-S/INC-USNC-URSI)*, pages 1641–1642, 2024.
- [66] Giovanni Simone Sticca, Memedhe Ibrahimi, Nicola Di Cicco, Francesco Musumeci, and Massimo Tornatore. On high-power optical amplification in hollow core fibers for energy efficiency and network throughput maximization. In *50th European Conference on Optical Communications (ECOC)*, 2024.
- [67] Nicola Di Cicco, Filippo Poltronieri, José Santos, Mattia Zaccarini, Mauro Tortonesi, Cesare Stefanelli, and Filip De Turck. Multi-objective scheduling and resource allocation of kubernetes replicas across the compute continuum. In *20th International Conference on Network and Service Management (CNSM)*, 2024.
- [68] Oleg Karandin, Aleix Lahoz, Nicola Di Cicco, Francesco Musumeci, and Massimo Tornatore. Resource-efficient implementation of multiple concurrent tree-based models in p4 switches using feature sharing. In *20th International Conference on Network and Service Management (CNSM)*, 2024.
- [69] Qiaolun Zhang, Nicola Di Cicco, Memedhe Ibrahimi, Nicola Di Cicco, Raul Almeida Júnior, Alberto Gatto, Raouf Boutaba, and Massimo Tornatore. Multi-

- objective scheduling and resource allocation of kubernetes replicas across the compute continuum. In *20th International Conference on Network and Service Management (CNSM)*, 2024.
- [70] Giovanni Simone Sticca, Memedhe Ibrahimi, Nicola Di Cicco, Francesco Musumeci, and Massimo Tornatore. Hollow core fiber as a long-term solution for capacity scaling in optical networks. In *2025 Optical Fiber Communication Conference and Exhibition (OFC)*, 2025.
- [71] José Santos, Mattia Zaccarini, Filippo Poltronieri, Mauro Tortonesi, Cesare Stefanelli, Nicola Di Cicco, and Filip De Turck. Hephaestusforge: Optimal microservice deployment across the compute continuum via reinforcement learning. *Future Generation Computer Systems*, 2025.
- [72] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2018.
- [73] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2016.
- [74] Maxime Gasse, Didier Chetelat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, 2019.
- [75] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21188–21198. Curran Associates, Inc., 2020.
- [76] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d’horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.
- [77] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.
- [78] André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. In *International Conference on Learning Representations*, 2022.

- [79] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. Learning to route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, page 185–191, 2017.
- [80] Xiaoliang Chen, Jiannan Guo, Zuqing Zhu, Roberto Proietti, Alberto Castro, and S. J. B. Yoo. Deep-rmsa: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks. In *2018 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3, 2018.
- [81] Muhammad Rehan Raza, Carlos Natalino, Peter Öhlen, Lena Wosinska, and Paolo Monti. Reinforcement learning for slicing in a 5g flexible ran. *Journal of Lightwave Technology*, pages 1–12, 06 2019.
- [82] Guillermo Bernárdez, José Suárez-Varela, Albert López, Bo Wu, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Is machine learning ready for traffic engineering optimization? In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11, 2021.
- [83] Sebastian Troia, Federico Sapienza, Leonardo Varé, and Guido Maier. On deep reinforcement learning for traffic engineering in sd-wan. *IEEE Journal on Selected Areas in Communications*, 39(7):2198–2212, 2021.
- [84] Guillermo Bernárdez, José Suárez-Varela, Albert López, Xiang Shi, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Magnneto: A graph neural network-based multi-agent system for traffic engineering. *IEEE Transactions on Cognitive Communications and Networking*, 9(2):494–506, 2023.
- [85] Chen-Yu Yen, Soheil Abbasloo, and H. Jonathan Chao. Computers can learn from the heuristic designs and master internet congestion control. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 255–274, New York, NY, USA, 2023. Association for Computing Machinery.
- [86] Piotr Gawłowicz and Anatolij Zubow. Ns-3 meets openai gym: The playground for machine learning in networking research. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 113–120, 2019.
- [87] Daniela M Casas-Velasco, Oscar Mauricio Caicedo Rendon, and Nelson LS da Fonseca. Intelligent routing based on reinforcement learning for software-defined networking. *IEEE Transactions on Network and Service Management*, 18(1):870–881, 2020.

- [88] Nan Geng, Tian Lan, Vaneet Aggarwal, Yuan Yang, and Mingwei Xu. A multi-agent reinforcement learning perspective on distributed traffic engineering. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–11, 2020.
- [89] Junjie Zhang, Minghao Ye, Zehua Guo, Chen-Yu Yen, and H Jonathan Chao. Cfr-rl: Traffic engineering with reinforcement learning in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2249–2259, 2020.
- [90] Chenyi Liu, Mingwei Xu, Yuan Yang, and Nan Geng. Drl-or: Deep reinforcement learning-based online routing for multi-type service requirements. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.
- [91] Bijoy Chand Chatterjee, Nityananda Sarma, and Eiji Oki. Routing and spectrum allocation in elastic optical networks: A tutorial. *IEEE Communications Surveys & Tutorials*, 17(3):1776–1800, 2015.
- [92] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.
- [93] Xiaoliang Chen, Baojia Li, Roberto Proietti, Hongbo Lu, Zuqing Zhu, and SJ Ben Yoo. Deeprmsa: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks. *Journal of Lightwave Technology*, 37(16):4155–4163, 2019.
- [94] Paul Almasan, José Suárez-Varela, Arnau Badia-Sampera, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: An optical network routing use case. *CoRR*, abs/1910.07421, 2019.
- [95] Carlos Natalino and Paolo Monti. The Optical RL-Gym: an open-source toolkit for applying reinforcement learning in optical networks. In *International Conference on Transparent Optical Networks (ICTON)*, page Mo.C1.1, July 2020.
- [96] Oleg Karandin, Francesco Musumeci, Omran Ayoub, Alessio Ferrari, Yvan Pointurier, and Massimo Tornatore. Quantifying resource savings from low-margin design in optical networks with probabilistic constellation shaping. In *2021 European Conference on Optical Communication (ECOC)*, pages 1–4, 2021.
- [97] Roberto Battiti and Giampietro Tecchiolli. The reactive tabu search. *ORSA journal on computing*, 6(2):126–140, 1994.
- [98] Roberto Battiti and Giampietro Tecchiolli. Training neural nets with the reactive tabu search. *IEEE transactions on neural networks*, 6(5):1185–1200, 1995.

- [99] Roberto Battiti et al. Reactive search: Toward self-tuning heuristics. *Modern heuristic search methods*, 4:61–83, 1996.
- [100] Roberto Battiti, Mauro Brunato, and Paolo Campigotto. Learning while optimizing an unknown fitness surface. In *International Conference on Learning and Intelligent Optimization*, pages 25–40. Springer, 2007.
- [101] Steven Prestwich. Tuning local search by average-reward reinforcement learning. In *International Conference on Learning and Intelligent Optimization*, pages 192–205. Springer, 2007.
- [102] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning, 2013.
- [103] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1889–1897. PMLR, 2015.
- [104] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv*, 06 2015.
- [105] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.
- [106] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *ECAI 2020*, pages 443–450. IOS Press, 2020.
- [107] Mirko Alicastro, Daniele Ferone, Paola Festa, Serena Fugaro, and Tommaso Pastore. A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems. *Computers & Operations Research*, 131, 2021.
- [108] Fuat Kosanoglu, Mahir Atmis, and Hasan Turan. A deep reinforcement learning assisted simulated annealing algorithm for a maintenance planning problem. *Annals of Operations Research*, 03 2022.
- [109] Alvaro H.C. Correia, Daniel E. Worrall, and Roberto Bondesan. Neural simulated annealing. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 4946–4962. PMLR, 25–27 Apr 2023.

- [110] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [111] Renaud Hartert, Pierre Schaus, Stefano Vissicchio, and Olivier Bonaventure. Solving segment routing problems with hybrid constraint programming techniques. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming*, pages 592–608, Cham, 2015. Springer International Publishing.
- [112] Renaud Hartert, Stefano Vissicchio, Pierre Schaus, Olivier Bonaventure, Clarence Filsfils, Thomas Telkamp, and Pierre Francois. A declarative and expressive approach to control forwarding paths in carrier-grade networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, page 15–28, 2015.
- [113] SIAE Microelettronica. <https://www.siaemic.com>, 2024. [Online; accessed 29-August-2024].
- [114] Hermann Wietgreffe et al. Using neural networks for alarm correlation in cellular phone networks. In *International Workshop on Applications of Neural Networks to Telecommunications (IWANNNT)*, pages 248–255. Citeseer, 1997.
- [115] Francesco Musumeci et al. Supervised and semi-supervised learning for failure identification in microwave networks. *IEEE Transactions on Network and Service Management*, 18(2):1934–1945, 2021.
- [116] Peter Szilagyi and Szabolcs Novaczki. An automatic detection and diagnosis framework for mobile communication systems. *IEEE Transactions on Network and Service Management*, 9(2):184–197, 2012.
- [117] Lujia Pan, Jianfeng Zhang, Patrick P.C. Lee, Marcus Kalandar, Junjian Ye, and Pinghui Wang. Proactive microwave link anomaly detection in cellular data networks. *Computer Networks*, 167:106969, 2020.
- [118] Omran Ayoub, Francesco Musumeci, Fatima Ezzeddine, Claudio Passera, and Massimo Tornatore. On using explainable artificial intelligence for failure identification in microwave networks. In *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, pages 48–55, 2022.
- [119] Tara Tandel, Omran Ayoub, Francesco Musumeci, Claudio Passera, and Massimo Tornatore. Federated-learning-assisted failure-cause identification in microwave networks. In *2022 12th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 1–7, 2022.

- [120] Francesco Lateano, Omran Ayoub, Francesco Musumeci, and Massimo Tornatore. Machine-learning-assisted failure prediction in microwave networks based on equipment alarms. In *2023 19th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 1–7, 2023.
- [121] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.
- [122] Kemal Davaslioglu and Yalin E. Sagduyu. Generative adversarial learning for spectrum sensing. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, 2018.
- [123] Bin Tang, Ya Tu, Zhaoyue Zhang, and Yun Lin. Digital signal modulation classification with data augmentation using generative adversarial nets in cognitive radio networks. *IEEE Access*, 6:15713–15722, 2018.
- [124] Yang Yang, Yang Li, Wuxiong Zhang, Fei Qin, Pengcheng Zhu, and Cheng-Xiang Wang. Generative-adversarial-network-based wireless channel modeling: Challenges and opportunities. *IEEE Communications Magazine*, 57(3):22–27, 2019.
- [125] Mohammad Nabati, Hojjat Navidan, Reza Shahbazian, Seyed Ali Ghorashi, and David Windridge. Using synthetic data to enhance the accuracy of fingerprint-based localization: A deep learning approach. *IEEE Sensors Letters*, 4(4):1–4, 2020.
- [126] Hojjat Navidan, Parisa Fard Moshiri, Mohammad Nabati, Reza Shahbazian, Seyed Ali Ghorashi, Vahid Shah-Mansouri, and David Windridge. Generative adversarial networks (gans) in networking: A comprehensive survey & evaluation. *Computer Networks*, 194:108149, 2021.
- [127] IV William H Clark, Steven Hauser, William C Headley, and Alan J Michaels. Training data augmentation for deep learning radio frequency systems. *The Journal of Defense Modeling and Simulation*, 18(3):217–237, 2021.
- [128] Marc Katzef, Andrew C. Cullen, Tansu Alpcan, and Christopher Leckie. Generative adversarial networks for anomaly detection on decentralised data. *Annual Reviews in Control*, 53:329–337, 2022.
- [129] Ender Ayanoglu, Kemal Davaslioglu, and Yalin E. Sagduyu. Machine learning in nextg networks via generative adversarial networks. *IEEE Transactions on Cognitive Communications and Networking*, 8(2):480–501, 2022.
- [130] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd*

International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014.

- [131] Yanqing Yang, Kangfeng Zheng, Chunhua Wu, and Yixian Yang. Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network. *Sensors*, 19(11), 2019.
- [132] Yanze Qu, Hailong Ma, Yiming Jiang, Liang Wang, and Jing Yu. A network data reinforcement method based on the multiclass variational autoencoder. *Security and Communication Networks*, 2022:1–10, 07 2022.
- [133] Mina Razghandi, Hao Zhou, Melike Erol-Kantarci, and Damla Turgut. Variational autoencoder generative adversarial network for synthetic data generation in smart home. In *ICC 2022 - IEEE International Conference on Communications*, pages 4781–4786, 2022.
- [134] Lareb Zar Khan, João Pedro, Nelson Costa, Antonio Napoli, and Nicola Sambo. Data augmentation to improve machine learning for optical network failure management. In *48th European Conference on Optical Communication (ECOC)*, 2022.
- [135] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.
- [136] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.
- [137] Qiang Xu and Rong Zheng. When data acquisition meets data analytics: A distributed active learning framework for optimal budgeted mobile crowdsensing. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.
- [138] Dario Azzimonti, Cristina Rottondi, and Massimo Tornatore. Reducing probes for quality of transmission estimation in optical networks with active learning. *Journal of Optical Communications and Networking*, 12(1):A38–A48, 2020.
- [139] Amin Shahraki, Mahmoud Abbasi, Amir Taherkordi, and Anca Delia Jurcut. Active learning for network traffic classification: A technical study. *IEEE Transactions on Cognitive Communications and Networking*, 8(1):422–439, 2022.

- [140] Ravid Shwartz-Ziv et al. Tabular data: Deep learning is not all you need. In *8th ICML Workshop on Automated Machine Learning*, 2021.
- [141] Andrey Malinin, Liudmila Prokhorenkova, and Aleksei Ustimenko. Uncertainty in gradient boosting via ensembles. In *International Conference on Learning Representations*, 2021.
- [142] Hadi Houssiany, Omran Ayoub, Cristina Rottondi, and Andrea Bianco. Using shap values to validate model’s uncertain decision for ml-based lightpath quality-of-transmission estimation. In *2023 23rd International Conference on Transparent Optical Networks (ICTON)*, pages 1–5. IEEE, 2023.
- [143] Omran Ayoub, Carlos Natalino, and Paolo Monti. Towards explainable reinforcement learning in optical networks: The rmsa use case. In *2024 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3. IEEE, 2024.
- [144] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.
- [145] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [146] Almuthanna Nassar and Yasin Yilmaz. Deep reinforcement learning for adaptive network slicing in 5g for intelligent vehicular systems and smart cities. *IEEE Internet of Things Journal*, 9(1):222–235, 2022.
- [147] Yifei Wei et al. Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor–critic deep reinforcement learning. *IEEE IoT Journal*, 6(2):2061–2073, 2018.
- [148] Yixuan Wang et al. Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. *IEEE Trans. Ind. Infor.*, 15(2):976–986, 2018.
- [149] Yaohua Sun et al. Deep reinforcement learning-based mode selection and resource management for green fog radio access networks. *IEEE IoT Journal*, 6(2):1960–1971, 2018.
- [150] Judy C Guevara et al. Qos-aware task scheduling based on reinforcement learning for

- the cloud-fog continuum. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 2328–2333. IEEE, 2022.
- [151] Seyedeh Negar Afrasiabi et al. Reinforcement learning-based optimization framework for application component migration in nfv cloud-fog environments. *IEEE Transactions on Network and Service Management*, 2022.
- [152] Yuhong Liu et al. A machine learning approach for task and resource allocation in mobile-edge computing-based networks. *IEEE Access*, 2020.
- [153] Lei Yang et al. Deep reinforcement learning for resource allocation in low latency edge computing networks. *IEEE Communications Letters*, 2020.
- [154] Kumar Narendra et al. Learning-based microservice placement and migration for multi-access edge computing. In *Proceedings of the IEEE International Conference on Cloud Computing*. IEEE, 2020.
- [155] Liang Zhao et al. When machine learning meets network management and orchestration in edge-based networking paradigms. *IEEE Communications Magazine*, 2020.
- [156] Priya Sharma et al. Deep reinforcement learning based mobile edge computing for iot. *IEEE Internet of Things Journal*, 2020.
- [157] Jin Lee et al. Deep reinforcement learning-based task scheduling in iot edge computing. *IEEE Transactions on Mobile Computing*, 2020.
- [158] Arun Kumar et al. Resource allocation with edge computing in iot networks via machine learning. *IEEE Internet of Things Journal*, 2020.
- [159] Xiaofei Chen et al. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa. *IEEE Transactions on Network Science and Engineering*, 2020.
- [160] Zhiwei Wei et al. Dynamic many-to-many task offloading in vehicular fog computing: A multi-agent drl approach. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 6301–6306. IEEE, 2022.
- [161] Zhen Gao et al. Fast adaptive task offloading and resource allocation via multi-agent reinforcement learning in heterogeneous vehicular fog computing. *IEEE Internet of Things Journal*, 2022.
- [162] Qi Chang et al. Cooperative edge caching via multi agent reinforcement learn-

- ing in fog radio access networks. In *ICC 2022-IEEE International Conference on Communications*, pages 3641–3646. IEEE, 2022.
- [163] Zhuo Li et al. Distributed edge computing offloading algorithm based on deep reinforcement learning. *IEEE Internet of Things Journal*, 2020.
- [164] Yu Zhang et al. Distributed machine learning for multiuser mobile edge computing systems. *IEEE Transactions on Vehicular Technology*, 2020.
- [165] Dan Wang et al. A cooperative computation offloading strategy with on-demand deployment of multi-uavs in uav-aided mobile edge computing. In *Proceedings of the IEEE International Conference on Computer Communications*. IEEE, 2020.
- [166] Jinglei Li et al. Multiobjective oriented task scheduling in heterogeneous mobile edge computing networks. *IEEE Transactions on Vehicular Technology*, 2022.
- [167] Lei Pan et al. A multi-objective clustering evolutionary algorithm for multi-workflow computation offloading in mobile edge computing. *IEEE Transactions on Cloud Computing*, 2023.
- [168] Arash Bozorgchenani et al. Multi-objective computation sharing in energy and delay constrained mobile edge computing environments. *IEEE Transactions on Mobile Computing*, 2021.
- [169] Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. Can the network be the ai accelerator? In *Proceedings of the 2018 Morning Workshop on In-Network Computing*, NetCompute '18, page 20–25, New York, NY, USA, 2018. Association for Computing Machinery.
- [170] Giuseppe Siracusano and Roberto Bifulco. In-network neural networks, 2018.
- [171] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Gianni Antichi, Paolo Costa, Hamed Haddadi, and Roberto Bifulco. Re-architecting traffic analysis with neural network interface cards. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 513–533, Renton, WA, April 2022. USENIX Association.
- [172] Tushar Swamy, Annus Zulfiqar, Luigi Nardi, Muhammad Shahbaz, and Kunle Olukotun. Homunculus: Auto-generating efficient data-plane ml pipelines for datacenter networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, page 329–342, New York, NY, USA, 2023. Association for Computing Machinery.

- [173] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pforest: In-network inference with random forests, 2022.
- [174] Changgang Zheng, Mingyuan Zang, Xinpeng Hong, Liam Perreault, Riyad Bensoussane, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. Planter: Rapid prototyping of in-network machine learning inference. *SIGCOMM Comput. Commun. Rev.*, 54(1):2–21, aug 2024.
- [175] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, jul 2014.
- [176] Pavol Mulinka and Pedro Casas. Adaptive network security through stream machine learning. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, SIGCOMM '18, page 4–5, New York, NY, USA, 2018. Association for Computing Machinery.
- [177] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. Matchmaker: Data drift mitigation in machine learning for large-scale systems. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 77–94, 2022.
- [178] Gemini Team et al. Gemini: A family of highly capable multimodal models, 2024.
- [179] Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024.
- [180] Abhimanyu Dubey et al. The llama 3 herd of models, 2024.
- [181] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.
- [182] Yujia Li et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, December 2022.
- [183] Daniel Adanza, Carlos Natalino, Lluís Gifre, Raul Muñoz, Pol Alemany, Paolo Monti, and Ricard Vilalta. Intentllm: An ai chatbot to create, find, and explain slice intents in teraflowsdn. In *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, pages 307–309, 2024.
- [184] Xiaotian Jiang, Min Zhang, Yuchen Song, Yao Zhang, Yidi Wang, Cheng Ju, and

- Danshi Wang. Opticomm-gpt: a gpt-based versatile research assistant for optical fiber communication systems. *Optics Express*, 32(12):20776–20796, 2024.
- [185] Yidi Wang, Chunyu Zhang, Jin Li, Yue Pang, Lifang Zhang, Min Zhang, and Danshi Wang. Alarmgpt: an intelligent alarm analyzer for optical networks using a generative pre-trained transformer. *Journal of Optical Communications and Networking*, 16(6):681–694, 2024.
- [186] Danshi Wang, Yidi Wang, Xiaotian Jiang, Yao Zhang, Yue Pang, and Min Zhang. When large language models meet optical networks: Paving the way for automation, 2024.
- [187] Changjie Wang, Mariano Scazzariello, Alireza Farshin, Simone Ferlin, Dejan Kostić, and Marco Chiesa. Netconfeval: Can llms facilitate network configuration? *Proceedings of the ACM on Networking*, 2(CoNEXT2):1–25, 2024.
- [188] Michele Bezzi. Large language models and security. *IEEE Security & Privacy*, 22(2):60–68, 2024.
- [189] Yibo Zhu et al. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review*, 45(4):523–536, August 2015.
- [190] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 202–215, New York, NY, USA, August 2016. Association for Computing Machinery.
- [191] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design guidelines for high performance rdma systems. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*, pages 437–450, USA, June 2016. USENIX Association.
- [192] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, et al. Empowering azure storage with {RDMA}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 49–67, 2023.

List of Acronyms

- AI** Artificial Intelligence
- AL** Active Learning
- ASAP** As-Soon-As-Possible
- ASIC** Application-Specific Integrated Circuit
- CP** Conformal Prediction
- DRL** Deep Reinforcement Learning
- GAN** Generative Adversarial Network
- GNN** Graph Neural Network
- IBN** Intent-Based Networking
- LLM** Large Language Model
- NIC** Network Interface Card
- NLP** Natural Language Processing
- ML** Machine Learning
- MLP** Multi-Layer Perceptron
- MORL** Multi-Objective Reinforcement Learning
- OR** Operations Research
- SA** Simulated Annealing
- SDN** Software-Defined Networking
- RL** Reinforcement Learning
- RWA** Routing and Wavelength Assignment
- VAE** Variational Autoencoder

VAP Venn-Abers Predictor

QoS Quality-of-Service

XAI Explainable Artificial Intelligence

Copyright Statement

In reference to IEEE copyrighted material which is used with permission in this Thesis, the IEEE does not endorse any of Politecnico di Milano's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

