Casetti, C.; Chiasserini, -C.F.; Marcato, S.; Puligheddu, C.; Mangues-Bafalluy, J.; Baranda, J.; Brenes, J.; Bocchi, F.; Landi, G.; Bakhshi, B. (2022). ML-driven Provisioning and Management of Vertical Services in Automated Cellular Networks. *IEEE Transactions on Network and Service Management* (Early Access).

DOI: https://doi.org/10.1109/TNSM.2022.3153087

# ML-driven Provisioning and Management of Vertical Services in Automated Cellular Networks

C. Casetti, *Senior, IEEE*, C.F. Chiasserini, *Fellow, IEEE*, S. Marcato, C. Puligheddu, *Member, IEEE*, J. Mangues-Bafalluy, J. Baranda, *Senior, IEEE*, J. Brenes, F. Bocchi, G. Landi, B. Bakhshi

***Abstract*—One of the main tasks of new-generation cellular networks is the support of the wide range of virtual services that may be requested by vertical industries, while fulfilling their diverse performance requirements. Such task is made even more challenging by the time-varying service and traffic demands, and the need for a fully-automated network orchestration and management to reduce the service operational costs incurred by the network provider. In this paper, we address these issues by proposing a softwarized 5G network architecture that realizes the concept of ML-as-a-Service (MLaaS) in a flexible and efficient manner. The designed MLaaS platform can provide the different entities of a MANO architecture with already-trained ML models, ready to be used for decision making. In particular, we show how our MLaaS platform enables the development of two ML-driven algorithms for, respectively, network slice subnet sharing and run-time service scaling. The proposed approach and solutions are implemented and validated through an experimental testbed in the case of three different services in the automotive domain, while their performance is assessed through simulation in a large-scale, real-world scenario. In-testbed validation shows that the use of the MLaaS platform within the designed architecture and the ML-driven decision-making processes entail a very limited time overhead, while simulation results highlight remarkable savings in operational costs, e.g., up to 40% reduction in CPU consumption and up to 30% reduction in the OPEX.**

***Index Terms*—5G platform, Vertical Services, SLA Management, ML-driven network management, Service Orchestration**

## I. Introduction

5G systems have been touted as capable of delivering an advanced platform primed to cater for vertical services, creating an ecosystem ripe for technical and business innovation. This crucial selling point for 5G has been addressed, in the years leading up to its commercial deployment, by several research projects attempting to identify challenges, problems and requirements of vertical industries that could be targeted by 5G capabilities. The '5G Promise' hinges, first and foremost, upon the ability to create an interface that effectively matches offer and demand between a network provider and a vertical, its customer. In other words, high-level Service Level Agreement (SLA) business requirements for the service instances that a vertical requests, must be mapped onto slice- and infrastructure-related requirements, which are reflected by the underlying network-level setup.

Among the projects [1], [2] that have addressed Machine Learning (ML) in the context of 5G networks, 5Growth [2] has sought to take this mapping one step forward, by applying ML capabilities to both the characterisation of the network context where traffic slices are deployed, and to the scaling of instantiated virtual services vis-a-vis unexpected surges in resource demand. To this end, we enhance the 5Growth architecture with an ML-as-a-Service (MLaaS) platform.

Leveraging a modular design, the MLaaS platform is equipped with a computing cluster that supports a large variety of ML models, which are uploaded to be trained and whose lifecycle is seamlessly managed. An interface with a monitoring platform collecting real-time data through Kafka feeds data to the model. Such ML models are then used for fully-automated service provisioning and management within the 5Growth architecture, a paradigm that is widely recognized as highly needed for 5G-and-beyond networks [3], [4]. In particular, looking at the virtual services requested by a vertical, we leverage the ML models to address two important challenges: (i) when and how to share network slice subnets among concurrent service instances, and (ii) when and how to scale such services, while accounting for both key performance requirements and OPEX.

We do so by providing the following main contributions:

- *Architectural design:* we define the internal architecture of the MLaaS platform, as well as enhancements to the 5Growth Vertical Slicer and Service Orchestrator entities, and the corresponding workflows, enabling the use of trained ML models for fully-automated service provisioning and management;
- *Algorithm design:* we introduce two ML-driven algorithms solving the problem of, respectively, network slice-subnet sharing [5] at the Vertical Slicer and run-time service scaling [6] at the Service Orchestrator. Both algorithms can swiftly adapt to time-varying load conditions, by leveraging the output of ML models to dynamically set their driving input parameters;
- *Experimental testbed validation and performance results:* we demonstrate the feasibility of our approach through a testbed implementing the whole 5Growth network architecture and the workflows between the aforementioned entities. Experimental results show the reduced impact of the ML-driven approach in terms of overall service instantiation and scaling time. Further, we assess the performance of the proposed ML-driven algorithmic so-

C. Casetti, C.F. Chiasserini, Silvio Marcato, and C. Puligheddu are with Politecnico di Torino, Italy, and CNIT, Italy. J. Mangues-Bafalluy, J. Baranda, and B. Bakhshi are with CTTC/CERCA, Spain. J. Brenes, F. Bocchi, and G. Landi are with Nextworks, Italy.

This work was supported by the EU Commission through the 5GROWTH project (Grant Agreement No. 856709), Spanish MINECO 5G-REFINE project (TEC2017-88373-R), and Generalitat de Catalunya 2017 SGR 1195.

lutions through simulations in a large-scale, real-world scenario, achieving up to 40% reduction of resource consumption in the case of slice-subnet sharing, and about 30% reduction of the OPEX in the case of service scaling.

As better discussed in Sec. VIII, the scope of our work and the solutions we present differ substantially from existing research on network slicing and resource allocation. Unlike previous work, we address *network slice-subnet sharing among different services*, rather than resource allocation sharing among different network slices. Moreover, while designing our service scaling scheme, we account for *both SLA violation costs and operational costs*, beside time-varying traffic load demands. In both cases, we design a novel architecture of the 5G network platform that can effectively leverage MLaaS for fully-automated service provisioning and management. Finally, as already mentioned, we provide a complete evaluation of the proposed framework that not only shows the performance of our solution, but also validates the interaction and functional synergy between the 5G architectural entities.

The rest of the paper is organized as follows. Sec. II introduces the recent 3GPP standards for slice management and describes how the 5Growth network architecture, including our MLaaS platform, provides a custom implementation of such standard specifications. Sec. III provides an overview of the proposed solutions for (i) network slice-subnet sharing and (ii) runtime slice adaptation to dynamic traffic conditions. The two solutions are then detailed in Sec. IV and Sec. V, respectively. Both experimental results obtained through the testbed we developed and simulation results in a large-scale scenario are shown in Sec. VII, for the automotive services described in Sec. VI. Finally, Sec. VIII discusses previous work, while Sec. IX draws some conclusions.

## II. Network Platform Architecture

This section presents some preliminaries on the 3GPP architecture for network slice management and orchestration, as well as for data analytics functions supporting closed-loop, cross-layer network control automation (Sec. II-A). This allows us to highlight how the design of the proposed MLaaS platform, and its interactions with other 5G network entities, is fully compliant with 3GPP standards. We then present the 5Growth architecture [2], [7], mapping its components into the relevant ones of the standard 5G Management System (Sec. II-B). Finally, we detail the architecture of our proposed MLaaS platform and how this addresses the need for ML models for fully automated service management, network orchestration, and resource control within the 5G network architecture (Sec. II-C).

### A. 3GPP management system and data analytics

Network slicing is one of the key features of 5G networks that allows creating multiple and concurrent logical networks, called network slices, over a shared physical infrastructure, each of them with its own key performance indicators (KPI) requirements, and security and isolation guarantees. An end-to-end 5G network slice spans both the radio access network and the 5G core network, and it includes a number of Network Functions (NF) which can be virtualized, deployed, orchestrated, and managed through the 5G Management System. As per 3GPP standards, a *network slice subnet* is a representation of the management aspects of a set of NFs managed through the 5G Management System and their required resources (e.g., compute, storage, and networking resources). These NFs, when virtualized, can be modelled as Virtual Network Functions (VNFs) and they can be combined together in an NFV Network Service (NFV-NS). In this sense, the NFV-NS provides the real implementation and deployment of a network slice subnet, i.e., a *network slice subnet instance (NSSI),* in the virtual infrastructure.

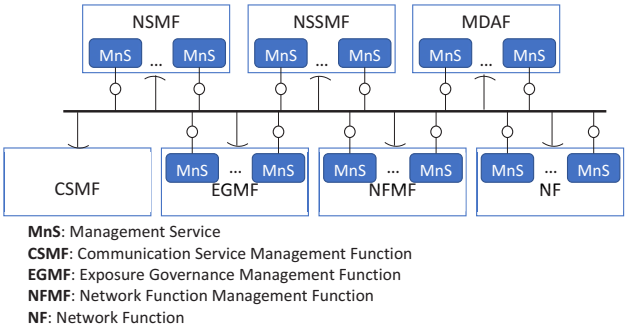



Fig. 1: 3GPP Service Based Management Architecture [8], including the MDAF block for fully-automated service management and resource orchestration.

The latest 3GPP standards [8] propose a Service Based Management Architecture (SBMA) for the 5G Management System (see Fig. 1), which includes Management Functions to deliver a variety of Management Services (MnS) to handle the management of single functions (NFMF), slices and communication services, as well as the exposure of the various services towards external entities (EGMF). In particular, the SBMA includes a Management Data Analytics Function (MDAF) to provide analytics services in support to automated network management and orchestration decisions. These decisions drive the logic of the Network Slice and Network Slice Subnet Management Functions (NSMF and NSSMF), which are in charge of handling the lifecycle of the 5G network slices and the orchestration of their resources across the various NSSI realizing an end-to-end slice.

At last, notice that the MDAF consumes monitoring data or records retrieved from the network and its management system, e.g., related to existing network slices and network service requests, or VNF performance, and yields analytics results to drive decisions related to the lifecycle management of network slices or the orchestration of network services.

### B. The 5Growth architecture: a custom implementation of the 3GPP management system

The 5Growth architecture, depicted in Fig. 2, is based on a hierarchy of functional elements operating at the different layers of a 5G network management system, from vertical service and network slice management, down to the orchestration of NFV services and infrastructure resources. They offer the man-

agement and orchestration (MANO) functionality [9] required to handle the lifecycle management of all architectural objects involved (i.e., vertical services, network slices, network slice subnets, NFV composite and nested network services, VNFs, and virtual links).

These management elements are assisted in their decisions and automated actions by an MLaaS platform (5Gr-MLaaSP), which is used to build ML models trained with multisource data collected through a cross-layer monitoring system. Thus, the 5Gr-MLaaSP prepares the models that drive the closed-loop actions taken at the different architectural layers on the basis of a multi-variable real-time context derived from the records maintained at each layer (e.g., for service demands, network slice instances, and their subnets) as well as from real-time monitoring data (e.g., on virtual resource consumption). In more details, the 5Growth architecture includes three functional elements: the Vertical Slicer (5Gr-VS), the Service Orchestrator (5Gr-SO), and the Resource Layer (5Gr-RL).

The 5Gr-VS handles the lifecycle of vertical services, their mapping into end-to-end network slices, and the provisioning and management of the slice subnets. The Vertical Service Management Function (VSMF) of the 5Gr-VS processes requests for Vertical Service Instances (VSI), defined through vertical service descriptors specifying the desired characteristics in terms of application-level parameters. On the basis of such characteristics, the VSMF identifies the Network Slice Instance (NSI) required to properly host the service, implementing the functionalities of an extended, vertical service-aware Communication Service Management Function (CSMF) within the 3GPP Management System. The provisioning of a network slice is handled by an enhanced NSMF embedded in the 5Gr-VS. This procedure involves the creation of the NFV-NSs implementing the NSSIs composing the end-to-end NSI associated with the VSI. These 5Gr-VS functionalities provide a concrete, custom and data-driven implementation of the 3GPP NSMF and NSSMF components, which coordinate the instantiation and deployment of the NFV-NS corresponding to NSSIs over an NFV MANO-like system represented by the 5Gr-SO. Each NFV-NS is deployed with the deployment flavour and *instantiation level (IL)* able to guarantee the vertical service requirements specified in the original request. Network slice subnets can be shared among multiple slice instances and, thus, among multiple service instances. Where needed, the 5Gr-VS may also trigger their scaling (i.e., modifying the IL of the corresponding NFV network services) to properly host additional vertical service instances. Sec. IV describes the ML-driven solution implemented to decide how to efficiently share network slice subnets among concurrent vertical services with different latency requirements.

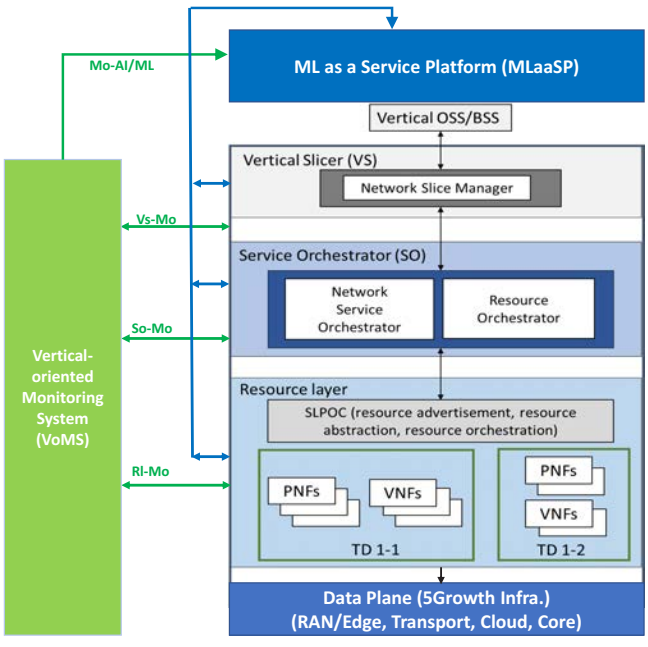



Fig. 2: The 5Growth architecture.

The 5Gr-SO handles the lifecycle management of NFV-NSs that build the slice subnets. For this purpose, it can handle both regular and composite NFV-NSs. It receives the NFV-NS requests from the 5Gr-VS and the available resources in the infrastructure from the 5Gr-RL (see below), and maps such requests over the infrastructure to fulfill their requirements, including sending requests for configuring virtual function interconnectivity through the transport network. In this direction, it coordinates the automated provisioning, monitoring, AIML model set up, and scaling of the virtual functions that compose the NFV-NS, according to model outputs.

The 5Gr-RL implements resource allocation operations in the underlying NFV infrastructure, abstracting the capabilities of the access, transport, and edge/cloud computing resources exposed to the 5Gr-SO. A Vertical-oriented Monitoring System (5Gr-VoMS) collects metrics and logs from these three functional elements, implementing a centralized and multi-layer monitoring platform. 5Gr-VoMS stores data related to the usage of physical and virtual infrastructure resources, to measure the performance of NSSIs or end-to-end network services, or service-level metrics collected from vertical applications. Monitoring data is used to feed the decision engines at each layer and, in particular, is used as input for the 5Gr-MLaaSP to build training datasets. The 5Gr-MLaaSP constitutes the 5Growth concrete implementation of the MDAF within the 3GPP Management System. More specifically, in this work, the 5Gr-MLaaSP is used to build trained ML models to support decisions about two orchestration actions performed at different levels of the 5Gr architecture, namely, network slice-subnet sharing and NFV-NS scaling at the 5Gr-VS and 5Gr-SO levels, respectively.

The multi-layer nature of the 5Growth architecture is fully compliant with the principles of the control loop applied at different layers of the 3GPP Management System [10], as represented in Fig. 3, with the mapping to the 5Growth components. In the 5Growth architecture, the 5Gr-VS and 5Gr-SO make decisions and enforce actions at the level of network slices and NFV network services, respectively, thus operating at the network slice level and at the network slice subnet level. In fact, in 5Growth the NSSIs are built and operated as NFV network services, where their internal lifecycle is managed directly by the 5Gr-SO, while their composition and sharing in end-to-end network slices is handled at the 5Gr-VS. Following this model, on one hand the 5Gr-VS makes

higher-level decisions that map the vertical service demand into network slices which can be decomposed in cross-service subnets shared among multiple slice instances. On the other hand, the lifecycle of the NFV network services building the single NSSIs is handled at the 5Gr-SO level, which makes decisions and enforces actions for their automated scaling. Both components are supported by the 5Gr-MLaaSP, which provides the data analytics functionalities. The 5Gr-MLaaSP builds datasets for training ML models using the multi-layer data collected by the 5Gr-VoMS. Such entity records statistics, metrics and KPIs at different layers, from single VNFs, e.g., in terms of consumed virtual resources or application indicators, up to NFV Network Services, Network Slices, and Vertical Services.

In summary, our architecture (i) translates vertical service requirements into network service requirements, and (ii) integrates ML in the management and orchestration workflows. In this sense, our approach is in accordance with the ETSI NFV guidelines, in which a generic architecture is devised to handle virtual services and virtual functions. In other words, the internal logic of the service should be handled by the service itself, while the architectural framework should deal only with services requirements that are expressed through generic parameters (e.g., virtual link bandwidth, number of CPUs, geographic location), and not through parameters that have to do with the specificity of each service. This makes the proposed architecture able to cope with any type of service.

At the same time, however, the architecture also enables decisions to be made depending upon the specific service. This is realized by downloading through the 5Gr-MLaaSP open API, and then by running, a trained ML model that is suitable for the service at hand. Indeed, the 5Gr-MLaaSP features a library of ML models that are fully aligned with the service offer of a given operator. Thus, by combining the generality of the architecture with a rich set of ML models, our framework can handle any kind of service, whilst also adapting to its specific operational goals.

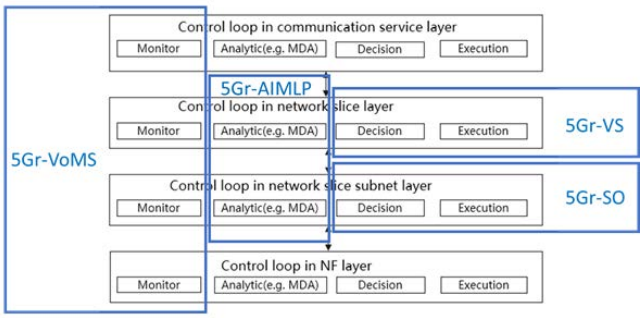



Fig. 3: Closed-loop control applied to different management system layers [10] and mapping onto the 5Growth architecture.

It is also worth noting that the 5Gr-MLaaSP design matches the most recent updates in the internal architectures of the 3GPP data analytics functions (NWDAF - the 5G Core Network function for data analytics - and MDAF). In particular, the old monolithic structure of the NWDAF is evolving in the latest versions of the 3GPP Rel. 17 specifications [11] to better fit the adoption of AI/ML techniques. Based on this model, the NWDAF is split into two different logic functions: the former is devoted to the training of ML models, with the capability of providing trained models towards external functions, while the latter implements the analytics service, e.g., performing inference and computing statistics or predictions. The NWDAF Analytics logical (AnLF) function acts as consumer of the NWDAF Model Training logical function (MTLF), exploiting its APIs for the discovery and exchange of trained models. A similar concept is applied in the design of the 5Gr-MLaaSP that implements the MTLF functionalities, while the decisions are delegated to other components of the 5Growth architecture that act as consumers of the 5Gr-MLaaSP services. In detail, the 5Gr-VS and 5Gr-SO are both in charge of retrieving the most suitable trained model from the 5Gr-MLaaSP (e.g., periodically or on-demand, based on the configured policies), which is then used to feed the analytics algorithms responsible for decision making. This approach fully decouples the training from the real-time analytics phase, allowing for autonomous updates of the trained models.

Importantly, there are various timescales at which models may be updated. And what is also relevant is when and how often models are actually used. As mentioned, the framework allows gathering data for training/updating the models that are already stored in the MLaaSP, and offering them to model consumers through open APIs. This allows, for instance, that every time a new service is instantiated, the updated version of the model is downloaded and run. Since dynamicity is also reflected in virtual services being continuously deployed and terminated, the operator benefits from such updates as new services are deployed.

### C. MLaaS for automated network management

The architecture and the fundamental workflow of the 5Gr-MLaaSP we designed and developed are depicted in Fig. 4, along with the other architectural entities with which the main interactions take place. The main components of the 5Gr-MLaaSP are as set forth below.

**External Interface**, through which an authorized external user can upload a model. The model may be already trained and onboarded, or it may still need to be trained, in which case, the user can provide a suitable dataset to be exploited for the training phase. In both cases, the user can specify (i) the scope of the model, i.e., the type of decision-making process to be used for (e.g., slice sharing, or service scaling), and (ii) the type of service the model/dataset should be used for (e.g., vehicle collision detection, digital twin). When the external user uploads a yet-to-be-trained model, it is the 5Gr-MLaaSP that takes care of the training and records the corresponding timestamp and, potentially, a validity time lapse. If no dataset is uploaded along with the model, the 5Gr-MLaaSP exploits the data collected through the 5Gr-VoMS platform about, e.g., the NSSIs or the network services performance. The configuration of the monitoring platform to gather the monitored data, aggregate it (e.g., through Apache Kafka), and feed it as input for real-time model execution can be properly set up. Models stored in the 5Gr-MLaaSP can be accessed by a 5Growth

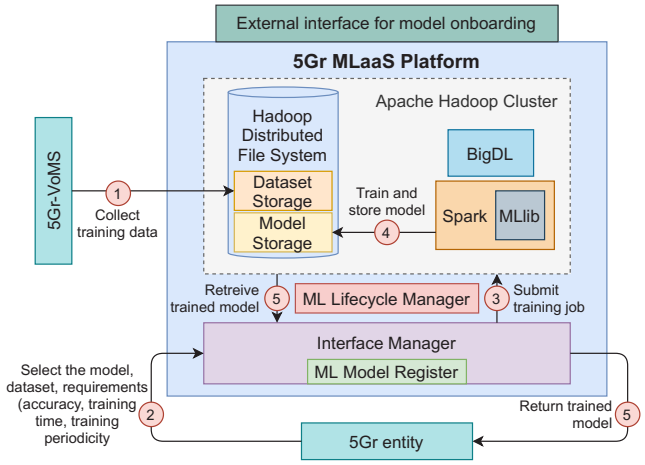



Fig. 4: 5Gr-MLaaSP architecture and its interaction with other architectural entities.

architecture entity through an open Representational State Transfer (REST) application programming interface (API).

**Model Register**, which records the models uploaded to the platform, their metadata, and pointers to the stored models and associated files.

**Lifecycle Manager**, which is in charge of the models lifecycle. Upon the uploading of a new model, it adds the corresponding entry to the Model Register and, if it is a yet-to-be-trained model, it triggers the training process using the appropriate AI/ML framework. After a model is trained, the Lifecycle Manager monitors its status: it can trigger a new training job either periodically, or whenever new data is available from the monitoring platform.

**Interface Manager**, which processes the requests for ML models coming from the architectural stack and forwards them to the proper block inside the computing cluster.

**Computing Cluster**, which is based on Apache Hadoop – one of the few enterprise-grade frameworks guaranteeing high efficiency, concurrency, reliability, and availability. It leverages Yet-Another-Resource-Negotiator (YARN) for the computing resources management, and the HadoopDistributed File System (HDFS) for the storage of datasets and models. The YARN cluster nodes have access to different ML frameworks, according to the requested model type. Apache Spark is used to train classic supervised and unsupervised models, while BigDL is used for Deep Neural Networks[1].

The envisioned workflow is as follows:

1) Monitoring data from the 5Gr-VoMS can be continuously collected, reformatted, and consolidated to build or update a training dataset. Training datasets are then saved in the HDFS Dataset Storage;
2) 5Gr-entities such as 5Gr-VS and 5Gr-SO trigger decision-making processes (e.g., slice-subnet sharing, service scaling) to support service lifecycle management operations (e.g., deployment, scaling) of a vertical service and its underlying network service. To enact such decision-making processes, the 5Gr-entity requests to the 5Gr-MLaaSP the available model catalogue that suits each of the problems at hand. The 5Gr-MLaaSP offers a set of available models, including already trained models as well as models that can be trained on-demand, as indexed by the Model Register. The 5Gr-entity selects the model, and may specify some requirements, e.g., accuracy, training time, or training periodicity so that the Lifecycle Manager can automatically keep the model fit;
3) In case the selected model requires to be trained, either because it has never been trained, its validity has expired, or it needs to be updated, a training job is submitted to YARN. If the requested model is ready to be used, it is directly fetched from the Model Storage;
4) Using the proper dataset from the Dataset Storage, the model is trained using either Spark MLlib, BigDL or Ray, depending upon the model type. The trained model is then saved in the HDFS Model Storage, while the ML Lifecycle Manager tracks the new trained model state and updates the Model Register accordingly;
5) The trained model is finally retrieved from the HDFS and returned to the requesting 5Gr-entity, which is responsible for its online execution.

We underline that, thanks to the ability to continuously collect data through the monitoring platform, the MLaaSP can update an ML model whenever necessary, or deemed useful.

## III. MLaaS for Automated Network Management in the 5Growth MANO Stack

To highlight how management and orchestration procedures in 5G networks can benefit from the MLaaS approach, we show how MLaaS is exploited at two different layers of the 5Growth stack, namely, the 5Gr-VS and the 5Gr-SO, to solve two different automated network management problems. This is enabled by the available 5Gr-MLaaSP open REST API, which allows consuming ML models from any external entity.

With regard to the first management problem, the 5Gr-VS is in charge of handling the vertical requirements, and in this sense, it deals with business relationships between the 5Growth provider and its customer (the vertical). This vertical service requirements are eventually translated into slice requirements by the 5Gr-VS, which is also in charge of generating the most efficient NFV-NS requests towards the 5Gr-SO based on the slice requirements. As detailed in Sec. IV, efficiency at this layer comes from NSSI sharing [5]. That is, if two slices have similar requirements and have part of the slice/service structure in common, instead of fully instantiating a new end-to-end slice, a slice subnet may be reused, with consequent resource savings.

In particular, we focus on latency requirements, hence NSSI sharing requires careful evaluation so that the target latency of both pre-existing and newly requested service instances can be met. This is the reason why the 5Gr-VS embeds an algorithm for classifying slice requests into latency classes as a function of the network context. However, dynamically adapting to such a context (e.g., to the traffic handled by the service entities) requires a careful definition of the latency classes, into which the requested services fall, that are eventually used by the

[1]Note that the MLaaSP could be extended to leverage also Ray (https://ray.io/), which can be used to pre-train reinforcement learning models.

aforementioned algorithm. This is the problem that the 5Gr-VS solves with the help of the ML model provided by the 5Gr-MLaaSP. In fact, at instantiation time, the 5Gr-VS requests, through the open API explained above, the previously-trained model, stored in the 5Gr-MLaaSP database. That is, the 5Gr-VS requests the model to solve the latency class definition problem. Once downloaded (together with the auxiliary code needed to run the model), the 5Gr-VS continuously runs the model to decide on the latency classes used depending upon the scenario conditions.

The second management problem that illustrates the flexibility of the MLaaS approach is still related to fulfilling the vertical SLA requirements, but at the 5Gr-SO, the entity that receives the requirements from the 5Gr-VS and matches them with the resources made available by the underlying infrastructure. As discussed in Sec. V, in this case the focus is on online scaling of nested NFV-NSs [6] based on actual operational data (not on requirements). As we go down the MANO stack, NSIs are requested in the form of composite NFV-NSs to the 5Gr-SO, and NSSIs hence become nested NFV-NSs.

Despite all the care taken when instantiating the service, there may be unexpected situations that create a sudden demand (e.g., entailing a sudden virtual CPU consumption increase) that could place SLA compliance at risk. Scaling is the solution we adopt to react to such operational events. In a general scenario, there are multiple factors affecting the scaling needs of virtual services. It depends upon the type of service and the instantiation level under execution, their latency and CPU requirements, as well as the available resources from the underlying infrastructure. An ML-driven approach helps leveraging all the relevant monitored data to make real-time automated decisions based on the best instantiation level (IL) that should be running at each instant to fulfill the service requirements based on the context in which the service is running. In this direction, at instantiation time the 5Gr-SO, in coordination with the 5Gr-VoMS, deploys the required probes to monitor the critical metrics. After that, the corresponding Kafka topics are created to gather such metrics and feed them to the ML model. Such an ML model follows exactly the same process as explained above, i.e., the 5Gr-SO requests to the 5Gr-MLaaSP the model for solving the scaling problem for the services that are being instantiated. Once it is downloaded, together with the auxiliary code, metrics are fed to the model, which is continuously run by the SLA manager module inside the 5Gr-SO to decide what is the correct IL for the running service. If such IL does not match the current one, a scaling operation of the nested NFV-NS is triggered if convenient, when both SLA violation costs and operating costs are accounted for.

## IV. ML-driven Slice-subnet Sharing for Efficient Service Provisioning

We now present an algorithmic solution for NSSI sharing, named slice-subnet sharing algorithm (SSA), which leverages an ML-driven parameter setting and is executed at the 5Gr-VS upon a new request made by a vertical for service instance deployment. After providing an overview of slice-subnet sharing at the 5Gr-VS (Sec. IV-A), we detail the SSA in Sec. IV-B, and describe how the SSA and the ML-driven configuration of the latency classes are integrated in the 5Gr-VS in Sec. IV-C.

### A. Slice-subnet sharing at the 5Gr-VS: An overview

To improve the efficiency of service deployment, it is of paramount importance to avoid the deployment of new, unnecessary NSSIs, when verticals request to the 5Gr-VS the deployment of service instances. Rather, already existing NSSIs should be reused whenever possible and allowed by isolation and KPI constraints, so that fewer virtual machines (VMs) are activated. On the other hand, sharing the same NSSI among services with different target latency may result in wasted computational capacity, and, thus, in higher operational costs. Indeed, when services with different latency constraints use the same NSSI, the most stringent constraint will have to be met also for the traffic associated with the least demanding service. This entails a waste of computing resources, which decreases with the increase in similarity among the values of the services target latency. Intuitively, under a low computing load, it is more beneficial to share NSSIs, even among services with quite different target latency, so as to fully utilize the already operating computing resources. On the contrary, as the computing load increases, only services with very similar target latency should share an NSSI, to avoid the waste of resources highlighted above. However, understanding the level of slice-subnet sharing that the 5Growth provider should allow under dynamic traffic and network conditions is a hard task.

We address this challenging issue by developing a slice-subnet sharing algorithm at the 5Gr-VS, which, as detailed in the next section, allows sharing an NSSI only if *possible* and *convenient*. In particular, it determines whether reusing an NSSI is beneficial based on whether services belong to the same *latency class*. Given an interval of possible target latency values, we define as latency classes the set of non-overlapping latency ranges, covering such interval. Clearly, the higher the number of latency classes, the smaller the latency range covered by each class.

It is easy to see that the set of latency classes to be used is the critical factor that drives the sharing algorithm: the narrower the latency classes, the more similar the target latency of services that can share an NSSI; instead, the broader the classes, the wider the difference in target latency of the services that can reuse the same NSSI. As detailed in Sec. IV-C, given the complexity of the problem, the time-varying system load, and the diverse types of services that the system has to deal with, we envision an ML-approach to determine the best set of latency classes and feed them as input to the SSA. As a result, the SSA is an ML-driven algorithm that leverages the output of a classification model as input to make slice-subnet sharing decisions.

### B. The slice-subnet sharing algorithm (SSA)

The SSA, presented in Algorithm 1, is executed at the 5Gr-VS every time a new service $s$ has to be instantiated. Beside

**Algorithm 1** Slice-subnet Sharing Algorithm (SSA)

**Require:** Latency classes, request $r = \langle \mathcal{V}_s, D_r^v, \lambda_r \rangle$, $\mathcal{R}$, $\theta_v$

1: $\mathcal{V}_r \leftarrow \mathcal{V}_s$ ▷ Given service request $r$, initialize $\mathcal{V}_r$ to the set of slice subnets composing the service
2: $\mathcal{O} \leftarrow \emptyset$ ▷ Initialize the output set $\mathcal{O}$ to empty set
3: **for all** $v \in \mathcal{V}_r$ **do**
4: $j^v \leftarrow$ `assign_latency_class`$(D_r^v)$ ▷ Determine the slice-subnet latency class
5: **for all** $\rho \in \mathcal{N}$ **do** ▷ For each running NSSI $\rho$ in $\mathcal{R}$
6: **if** ($\rho$ implements $v \in \mathcal{V}_r$) $\wedge$ $j^v = j^\rho$ **then** ▷ Check if the NSSI implements a slice subnet in $\mathcal{V}_r$ and its latency class
7: **if** $\theta_v[\Lambda(\rho) + \lambda_r] + \frac{1}{\min_{\hat{r}} D_{\hat{r}}^\rho} \leq \bar{\mu}_\rho$ **then**
8: $\mu_\rho \leftarrow \theta_v[\Lambda(\rho) + \lambda_r] + \frac{1}{\min_{\hat{r}} D_{\hat{r}}^\rho}$ ▷ Adjust capability of the NSSI
9: $\mathcal{O} \leftarrow \mathcal{O} \cup (\rho, \mu_\rho)$
10: $\mathcal{V}_r \leftarrow \mathcal{V}_r \setminus v$
11: **if** $\mathcal{V}_r = \emptyset$ **then** ▷ Check if all the slice subnets are instantiated
12: **break**
13: **if** $\mathcal{V}_r \neq \emptyset$ **then** ▷ If still slice subnets to instantiate
14: **for all** $v \in \mathcal{V}_r$ **do**
15: $\rho \leftarrow$ `create_NSSI`$(v)$
16: $\mu_\rho \leftarrow \theta_v \lambda_r + \frac{1}{D_v^r}$
17: $\mathcal{O} \leftarrow \mathcal{O} \cup \{\langle \rho, \mu_\rho \rangle\}$
18: **return** $\mathcal{O}$

the latency classes configuration, the algorithm takes as input: (i) the newly requested service instance $r$, along with the set $\mathcal{V}_s$ of slice subnets $v$ composing the service, and the expected service traffic load $\lambda_r$; (ii) the target latency, $D_r^v$, associated with each slice subnet $v$, its complexity factor $\theta_v$ indicating the amount of virtual CPU (vCPU) required by $v$ to process a traffic unit; and (iii) the maximum computing capability, $\bar{\mu}_\rho$, that can be allocated to an existing NSSI, $\rho \in \mathcal{R}$. Then the algorithm initializes two sets: $\mathcal{V}_r$ to set $\mathcal{V}_s$ and the SSA output, $\mathcal{O}$, to the empty set (Lines 1–2). $\mathcal{O}$ will eventually include tuples composed of two elements: the NSSI identifier and the amount of computing resources assigned to the NSSI. Once identified the latency class of each slice subnet $v$ in $\mathcal{V}_r$ (Line 4), the algorithm looks for NSSIs already instantiated that can be reused for the deployment of service request $r$. In particular, the SSA determines whether any of the current NSSIs, $\rho \in \mathcal{R}$, can be shared with the new service, i.e., whether it implements a slice subnet that is included in $\mathcal{V}_r$ and falls in the same latency class as the slice subnet in $\mathcal{V}_r$ (Lines 5–6).

For each shareable NSSI, $\rho$, identified by the SSA, the computing capability, $\mu_\rho$, may need to be adjusted based on the current load of the NSSI (i.e., $\Lambda(\rho)$) and the additional load associated with the newly requested service instance (i.e., $\lambda_r$), till the maximum value $\bar{\mu}_\rho$. This is done by modeling the processing time of a VM through an M/M/1 queue, as done in [12], [13], [14], [15], [16] (Line 8). The tuple ⟨NSSI id, computing allocation⟩, i.e., $< \rho, \mu_\rho >$, is then added to the output set (Line 9) and $v$ is removed from the set $\mathcal{V}_r$ of slice subnets to instantiate. The sharing process ends when either all existing NSSIs have been processed or all slice subnets have been instantiated, i.e., $\mathcal{V}_r$ becomes empty (Line 11). Finally, if some slice subnets cannot share any existing NSSI, new NSSIs are created, and the necessary computing resources are allocated (Lines 13–16).

### C. ML-driven SSA parameter setting

To determine the best set of latency classes to be fed to the SSA, the 5Gr-VS interacts with the 5Gr-MLaaSP, performing the steps depicted in Fig. 5 and detailed below:

1) The current number of deployed NSSIs and the resource utilization metrics (e.g., vCPU consumption) are fetched continuously from the 5Gr-VS Catalog and the monitoring platform, so as to build datasets that can be used for updating the ML model;
2) the ML model is trained and stored;
3) the 5Gr-VS Arbitrator block requests the trained model to the 5Gr-MLaaSP to be used for determining the SSA input parameters when needed;
4) upon receiving a request for service instance deployment, the VSI/NSI Coordinator within the 5Gr-VS passes the request to the Arbitrator;
5) the Arbitrator retrieves the number of currently deployed NSSIs from the 5Gr-VS Catalog;
6) it then executes the trained ML model to determine the latency classes and pass them to the SSA (running at the Arbitrator);
7) the SSA determines which NSSI(s) can be shared and whether the computing resources assigned to an NSSI need to be adjusted; it then provides this information to the VSI/NSI Coordinator, which triggers the service instantiation process at the 5Gr-SO.

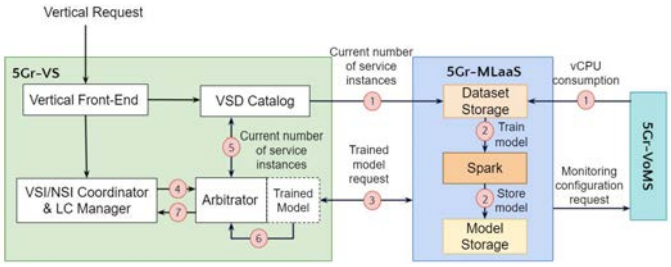



Fig. 5: Interaction between 5Gr-VS and 5Gr-MLaaSP, and 5Gr-VS internal structure.

## V. ML-driven Service Scaling for SLA Management and OPEX Minimization

In the previous section, slice-subnet sharing has been discussed as a use case of ML-driven provisioning of vertical slices, exploiting the interaction between 5Gr-MLaaSP and 5Gr-VS. We now focus on the 5Gr-SO, and its interaction with the 5Gr-MLaaSP, for an ML-driven SLA management and OPEX minimization. After providing in Sec. V-A an overview of the proposed approach, we detail our algorithmic solution in Sec. V-B, and the design for ML-driven operations at the 5Gr-SO in Sec. V-C.

### A. Service scaling at the 5Gr-SO: An overview

The ultimate goal of the 5G network provider is to maximize the profit via minimizing the operational expenditure (OPEX), which is mainly due to:

- The cost of NSIs (instantiated as NFV-NSs at the 5Gr-SO), such as VNF license cost;
- The cost of operating NSI/NFV-NSs, i.e., the cost of keeping instances up and running, such as energy cost;
- The SLA violation cost, i.e., the penalty that the provider should pay if the maximum value of the target metric (in this case, latency), upon which the provider and the vertical agreed, is violated.

The challenges for minimizing OPEX are twofold. First, these costs are conflicting; to minimize the SLA violation cost, the provider should allocate sufficiently large resources to handle the peak load of the service, which would significantly increase the instantiation and operation cost. Vice versa, if the provider aimed to minimize the provisioning and operation cost by allocating the minimum resources to the services, it would incur a considerable SLA violation cost. Thus, the problem consists in finding the optimal trade-off between the costs, i.e., the *optimum IL* that minimizes the OPEX. The second challenge is that the optimum IL depends upon the traffic load, which is time-varying, and it is a non-trivial task to understand when *scaling out/in* should be performed and to which IL, so as to avoid SLA violation costs. To address this latter point, we leverage an ML-based approach and, as detailed in Sec. V-C, we design the internal architecture of the 5Gr-SO to accommodate ML-driven operations in an effective and efficient manner.

### B. NFV-NS resource scaling algorithm

The resource scaling algorithm is a logic in the SLA Manager of the 5Gr-SO, which uses an ML model, already trained and maintained by 5Gr-MLaaSP, to determine the suitable IL of the NFV-NS.

Let us first introduce the mathematical expression for the aforementioned costs and the problem we address. We denote the NFV-NS latency threshold and its associated penalty, as specified in the SLA, by $\delta$ and $p$, respectively. Let $\tau$ be the service latency; we define the cost for violating the SLA as:

$$\sigma_{\mathsf{sla}} = p(\tau - \delta)\,. \tag{1}$$

The NFV-NSs are composed of a number of VNFs, but there is typically a bottleneck VNF that dominates the service latency, as also better highlighted in Sec. VI. Thus, to comply with the SLA, the provider needs to scale in/out the bottleneck VNF. The descriptor of the NFV-NS, specifies a set $\mathcal{L}$ of ILs where $\forall l \in \mathcal{L}$ corresponds to $n_l$ instances of the bottleneck VNF. The cost of creating a new instance of the VNF is $\sigma_{\mathsf{ins}}$, and the cost of using an instance per unit of time is $\sigma_{\mathsf{opr}}$. There is no cost for termination.

Given a time period $\mathcal{T}$, let $\mathcal{N}$ be the set of NFV-NS requests arriving in this period, and $\mathcal{C}$ be the set of instances created during this period; moreover, let $t_c^i$ and $t_t^i$ be, respectively, the instantiation and termination times of instance $i$. The objective is to minimize the OPEX, defined as

$$OPEX = \sum_{r\in\mathcal{N}} \sigma^r_{\mathsf{sla}} + \sum_{i\in\mathcal{C}} \big(\sigma_{\mathsf{ins}} + \sigma_{\mathsf{opr}}(t_t^i - t_c^i)\big). \tag{2}$$

As mentioned, to achieve such a goal, the network provider needs to find the optimum IL to be applied at each time $t \in \mathcal{T}$.

To this end, we adopt the following approach. We first determine the required IL to not violate the SLA through an ML model that has been trained within the 5Gr-MLaaSP and delivered to the 5Gr-SO. The 5Gr-MLaaSP trains the ML model using a dataset where the features are (i) average CPU utilization over the active instances $u_{\mathsf{cpu}}$, (ii) average memory utilization $u_{\mathsf{ram}}$, (iii) service latency $\tau$, and (iv) current IL $l$; the label is the target IL to satisfy the target latency. Then, in the operation phase, the following steps are performed at every period $j$:

1) The monitoring data $u^j_{\mathsf{cpu}}$, $u^j_{\mathsf{ram}}$, $\tau^j$, and $l^j$ are collected;
2) The exponential moving averages of the monitoring data are updated as $\bar{x} \leftarrow \alpha x^j + (1-\alpha)\bar{x}$;
3) The ML model runs using the averages $\bar{u}^j_{\mathsf{cpu}}$, $\bar{u}^j_{\mathsf{ram}}$, $\bar{\tau}$, and $l^j$, and provides $l^{j+1}_{\mathsf{ml}}$ so as to comply with the SLA (but without necessarily minimizing the OPEX);
4) The ML-Driven Service Resource Scaling (ML-RS) algorithm runs using $l^{j+1}_{\mathsf{ml}}$ and determines $l^{j+1}$ that yields the best trade-off between the costs;
5) If $l^j \neq l^{j+1}$, the SLA Manager triggers the scaling operation to deploy the new IL.

Though fast switching between ILs can decrease $\sum_{r\in\mathcal{N}} \sigma^r_{\mathsf{sla}}$ and the operation cost, i.e., $\sum_{i\in\mathcal{C}} \big(\sigma_{opr}(t_t^i - t_c^i)\big)$, it incurs significant instantiation cost $\sum_{i\in\mathcal{C}} \sigma_{\mathsf{ins}}$. To alleviate it, three steps are taken into account in this solution. First, the monitoring period is large enough to avoid triggering the scaling operation per request. Second, instead of instantaneous monitoring data, we use the exponential moving average as input to the model. Third, the ML-RS algorithm, presented in Algorithm 2, takes into account the SLA violation cost, instantiation cost, and operation cost to obtain the target IL.

This algorithm, in addition to the IL suggested by the ML model, $l^{j+1}_{\mathsf{ml}}$, takes the SLA violation and operation costs in this monitoring interval, which are respectively denoted by $\Sigma^j_{\mathsf{sla}}$ and $\Sigma^j_{\mathsf{opr}}$. It also takes the scaling direction $SD^{j-1}$ suggested by the ML model in the previous interval where $SD = 0$ implies no scaling, and $SD > 0$ ($SD < 0$) means scaling out (in). The algorithm at the beginning (Line 1) finds the scaling direction suggested by the ML model in this interval. If it is "scale out" but the model has changed the direction, $SD^{j-1} \leq 0$, the SLA violation cost of this interval is saved as the accumulated SLA violation cost $\Sigma_{\mathsf{sla}}$ (Line 4). However, if the model is continuously requesting to scale out, the accumulated SLA violation cost is updated and, if it is large enough that implies it is beneficial to create new instances to decrease the violation cost, then, the suggested IL is selected (Line 10). In a similar way, when the model suggests scaling in (Line 11), the algorithm decides to scale in only if the previous suggestion was also scaling in and the operation cost is large enough.

**Algorithm 2** ML-Driven Service Resource Scaling (ML-RS)

**Require:** $l_{\mathsf{ml}}^{j+1}, \Sigma_{\mathsf{sla}}^{j}, \Sigma_{\mathsf{opr}}^{j}, SD^{j-1}$
1: $SD^{j} \leftarrow \text{sign}(n_{l^{j}} - n_{l_{\mathsf{ml}}^{j+1}})$
2: **if** $SD^{j} > 0$ **then** ▷ scaling out suggestion by ML model
3: **if** $SD^{j-1} \leq 0$ **then** ▷ a new scaling out suggestion
4: $\Sigma_{\mathsf{sla}} \leftarrow \Sigma_{\mathsf{sla}}^{j}$
5: **else** ▷ ML model keeps requesting scaling out
6: $\Sigma_{\mathsf{sla}} \leftarrow \Sigma_{\mathsf{sla}} + \Sigma_{\mathsf{sla}}^{j}$
7: **if** $\Sigma_{\mathsf{sla}} > \beta\sigma_{\mathsf{ins}}$ **then** ▷ large SLA violation cost
8: $l^{j+1} \leftarrow l_{\mathsf{ml}}^{j+1}$
9: **else**
10: $l^{j+1} \leftarrow l^{j}$
11: **else if** $SD^{j} < 0$ **then** ▷ scaling in suggestion by ML
12: **if** $SD^{j-1} \geq 0$ **then** ▷ a new scaling in suggestion
13: $\Sigma_{\mathsf{opr}} \leftarrow \Sigma_{\mathsf{opr}}^{j}$
14: **else** ▷ ML model keeps requesting scaling in
15: $\Sigma_{\mathsf{opr}} \leftarrow \Sigma_{\mathsf{opr}} + \Sigma_{\mathsf{opr}}^{j}$
16: **if** $\Sigma_{\mathsf{opr}} > \gamma\sigma_{\mathsf{ins}}$ **then** ▷ large operation cost
17: $l^{j+1} \leftarrow l_{\mathsf{ml}}^{j+1}$
18: **else**
19: $l^{j+1} \leftarrow l^{j}$
20: **return** $l^{j+1}$

The conditions in Lines 7 and 16 aim to let the trade-off between the costs, where $0 < \beta < 1$ and $0 < \gamma < 1$ are tunable parameters, depend upon the dynamics of the traffic load. The higher the traffic load dynamic, the larger these parameters to avoid too many instantiations and terminations by small changes in the traffic load. However, if traffic load changes smoothly, the value of the parameters can be small to minimize the SLA violation cost and the operation cost.

### C. ML-driven 5Gr-SO design

In accordance with the discussions in Sec. II, when instantiating an NFV-NS, the workflow describing the interaction between the 5Gr-SO and 5Gr-MLaaSP follows a model in which the MTLF and AnLF are located in different architectural entities. More specifically, the 5Gr-MLaaSP trains the model based on the previously uploaded dataset and makes it available to external entities. The 5Gr-SO downloads the model and continuously runs it during the lifetime of the service for which scaling decisions are needed.

This high-level architectural idea requires multiple steps to be deployed, which are mainly related to the configuration of a complete data engineering pipeline, since monitoring job configuration and data collection until execution of the management and orchestration procedure based on the decision made by the model. Thus, once the complete pipeline is in place and integrated with the 5Gr-SO operation, the closed-loop automated network management decisions can be made, since relevant metrics are gathered and ingested by the ML model, which infers the best possible IL, which in turn generates the corresponding scaling procedure, when needed.

More specifically, the workflow is as follows (see [17] for further details). At the end of the NFV-NS instantiation process, the 5Gr-SO requests the 5Gr-VoMS to configure the monitoring jobs for the service, as specified in its network service descriptor (NSD). If the NSD also embeds an AI/ML information element (IE) for solving a specific problem (in this case, scaling), the SLA manager inside the 5Gr-SO detects it and starts the configuration of the data engineering pipeline.

First, a dedicated Kafka topic is created. Second, data scrapers are also created together with the 5Gr-VoMS that filter the relevant information to be fed to the scaling topic. After that, the model is downloaded from the 5Gr-MLaaSP/MTLF, and the SLA manager creates an Apache Spark streaming job in charge of feeding the ML model with real-time information (including the current IL) for scaling decision making (AnLF). If the inferred IL by the model is different from the current one, a scaling procedure requesting the change of the IL is triggered. Furthermore, during scaling procedure execution, the model inference process is stopped to avoid unexpected transient effects and is created again for the scaled service at the end of the scaling process.

It is worth mentioning that in the current system design, the model does not need to be changed after a scaling operation [2]. Indeed, all possible states (i.e., ILs) of the service are considered during the training phase, and the monitored metric values related to different instances are averaged during inference so that a single model can be used regardless of the current IL[3]. The current approach has the advantage of scaling very well with the number of possible ILs. On the contrary, handling multiple ML models as the value of service IL changes would require the download from the MLaaSP of as many trained ML models as the number of ILs allowed for the service, with only one specific model being used at a time according to the current IL value.

## VI. Automotive Services

To validate our approach to network slice provisioning and management, we take as reference services three relevant use cases in the automotive domain, namely, (i) vehicle collision detection at intersections (CD), (ii) see-through (ST), and (iii) destination-aware bird-eye view (DaBEV). The first two are representative of safety services, while the third is an example of convenience services for vehicular users [18]. The three services are depicted in Fig. 6 and detailed below.

**Collision detection.** The CD service detects vehicles on collision course and sends them an alert message. It exploits two types of messages defined by ETSI: Cooperative Awareness Messages (CAMs), which are periodically transmitted by vehicles and carry the position, speed, acceleration, and heading of the sender, and the Decentralized Environmental Notification Messages (DENMs), which are delivered to vehicles to notify them about events or dangerous situations. The service includes the following VNFs:

the *Cooperative Infrastructure Manager (CIM)*, which receives, decodes, and stores CAMs sent by the vehicles within the area covered by the CD service (see step 1 in Fig. 6(left));

[2]It is assumed that the service implementation provides the corresponding logic (e.g., load-balancing) to effectively support the scaling operation.

[3]Note, however, that the current IL is an input to the model during inference.

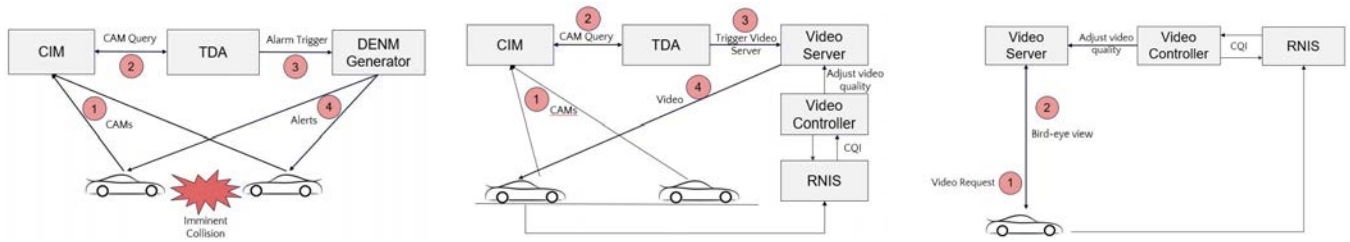



Fig. 6: Service structure: collision detection (left), see through (center), and destination-aware bird-eye view (right).

*the Trajectory and Detection Algorithm (TDA)*, which queries the CIMs for new CAMs (step 2) and runs a trajectory-based algorithm (i.e., the one presented in [19] and evaluated in [20]), to detect pairs of vehicles on collision course;

the *DENM Generator*, which, triggered by the TDA (step 3), encodes and sends (unicast) alarm messages to the vehicles detected to be on collision course (step 4), so that, e.g., the emergency braking system aboard vehicles can be activated.

Upon requesting a CD service instance, the vertical specifies the geographical area (e.g., set of intersections) that has to be covered and the estimated number of users to serve. Also, since the CD should be combined with other collision avoidance mechanisms based on physical sensors aboard the vehicles, the maximum target CD latency specified by the vertical is set to 20 ms [21]. Notice that the dominant contribution to the service processing time is due to the TDA, which is thus the botteneck VNF of the CD service.

**See-through.** It provides a real-time view of the surrounding area of the requester, to avoid collisions when an overtaking manoeuvre is executed. The service includes:

the *CIM* and the *TDA*, which, as above, store the vehicle's CAMs (step 1 in Fig. 6(center)) and run the trajectory-based algorithm (step 2), respectively. Upon detecting pairs of vehicles that would be on collision course if one of them moved to the left lane, the TDA triggers the Video Server (step 3);

the *Video Server*, which fetches the video from the smart-city cameras providing the best view of the surroundings of the tagged vehicles, and encodes and transmits video frames to the vehicles (step 4);

the *Video Controller*, which reports periodically to the video server the video quality to be used, based on the channel quality experienced by the vehicles;

the *Radio Network Information Service (RNIS)*, which is co-located with the radio point of access and exposes radio contextual information, namely, the Channel Quality Indicator (CQI); unlike the above functions, it is implemented as a physical network function (PNF).

The quality of the video should be high enough to guarantee at least a rate of 30 frames per second (fps), i.e., a good representation of the position and movements of the involved vehicles, sacrificing the resolution if necessary and maintaining the maximum latency below 150 ms [22]. In the ST service, it is the Video Server that exhibits the highest level of complexity, followed by the TDA.

**Destination-aware bird-eye view.** It provides a real-time view of the area between the requester and its intended destination, leveraging smart-city cameras located along the vehicle's route. It includes:

the *Video Server*, receiving the vehicle request (step 1), and fetching the smart-city videos, encoding them, and transmitting the bird-eye view to the requester (step 2);

the *Video Controller*, providing, as above, the Video Server with the video quality to be used;

the *RNIS*, providing the video controller with the vehicles' CQI; as before, it is implemented as a PNF.

The ideal output should be a high resolution (e.g., $1920 \times 1080$) and low fps (e.g., 7 fps), since this is a convenience service and does not have to provide the user with a highly dynamic context. For each received request, the processing load on the Video Server (which is the dominant VNF here) depends upon the number of input video cameras from which a stream is required to represent the bird-eye view of the area of interest. The maximum service latency is set to 1 s.

Given the above services, we consider that CIM and TDA VNFs form the *Trajectory Detection Slice Subnet (TDSS)*, while Video Server and Video Controller compose the *Adaptive Video Slice Subnet (AVSS)*. Finally, we remark that all the above services require the use of a radio access network for vehicle-to-infrastructure connectivity.

## VII. Validation and Performance Evaluation

In this section, we first detail the real-world scenario we used to build our training datasets, and to derive large-scale simulation results (Sec. VII-A). Then, through our experimental testbed, we validate the interaction between the 5Growth entities (5Gr-VS and 5Gr-SO) and the 5Gr-MLaaSP, as well as the proposed ML-driven algorithms for slice-subnet sharing at the 5Gr-VS and run-time scaling at the 5Gr-SO, in a small-scale scenario (Sec. VII-B). Finally, we show the performance results of both solutions, via simulation in the aforementioned large-scale, real-world scenario (Sec. VII-C).

### A. Large-scale reference scenario, datasets, and ML model

We consider an 11 $km^2$ area of the city of Turin, Italy, comprising 24 major crossroads and a total of 112-km road stretches. Vehicle mobility is simulated thanks to the Simulation of Urban Mobility (SUMO) and the Turin SUMO traffic (TuST) trace [23]. As the latter refers to 24-hour traffic in a weekday, we select 6 different time slots that are representative of different vehicle densities during the day; the 14 metropolitan zones included in the trace are depicted in Fig. 7. Also, we consider the Metro Node of the cellular network as the point of presence (PoP) where service instances should be deployed. The number of vehicles travelling on the whole area varies

from 2,110 in the 3am–4am time slot to 32,117 in the 6pm–7pm time slot.

In the scenario under study, a different number of CD instances are requested by the automotive vertical, depending upon the vehicle density in the considered time slots, and the CD service is provided to all vehicles entering an intersection covered by such service. Tab. I reports the relation between the average number of vehicles per km in the overall geographical area, and the number of CD instances that have to be deployed. Unless otherwise specified, upon entering the area, 50% of the vehicles request the ST service, while 30% request the DaBEV service, and the arrival of requests for these services is modeled as a Poisson process with a rate value set as in Tab. I. The lifetime of the ST and DaBEV is set according to the time taken by the vehicles to travel across the urban area. Finally, we consider that each VNF is implemented in a VM, and each active VM can use up to 8 vCPUs.

TABLE I: CD instances in the considered area

| Veh. density [veh./km] | No. of CD instances | Vehicle rate [veh./s] |
|---|---|---|
| 1.10 | 1 | 0.59 |
| 3.04 | 3 | 2.24 |
| 8.36 | 8 | 6.79 |
| 11.04 | 9 | 7.11 |
| 12.11 | 10 | 8.09 |
| 15.38 | 11 | 8.92 |

The datasets[4] used to train the ML models within the 5Gr-MLaaSP have been obtained through extensive simulations in the above real-world scenario for the slice-subnet sharing problem, as well as through experimental tests on an enhanced version of the testbed in [21] for service scaling. Specifically, the latter tests have been leveraged to assess the computing requirements of the considered VNFs. For slice-subnet sharing, each simulation is performed for a given latency class configuration, among the many that we have considered, and vertical's request arrival rate. Such data is then labeled so as to identify the best configuration with respect to the CPU consumption. For service scaling, we experimentally derived three datasets, one for each of the considered automotive services. The testbed has been configured to run the CIM, TDA, Video Server, and Video Controller in the machine accommodating the Edge Host, which also runs a srsRAN virtualized LTE eNB. A second machine, which hosts the UE, connects to the Edge Host through the LTE link and acts as a client generating requests to the offered automotive services. The rate of such requests varies over time to emulate the behaviour of multiple users. The CPU consumption is monitored at runtime, while the end-to-end latency is computed offline by post-processing the experiment logs. The final datasets are built adding as label the IL that can successfully handle the computational load associated with each rate of user requests. We considered that each service has two possibile ILs, hence the dataset (and the NSD) includes two labels: "small IL" and "big IL". To label the datasets, we set the target processing times for the considered services to values that, based on our testbed experiments, allow meeting the maximum latency reported in Sec. VI.

[4]The datasets will be made publicly available on github upon paper acceptance.

Fig. 7: Large-scale scenario: Turin metropolitan area.

To perform the prediction of the best parameter values to use within the SSA and the the ML-RS algorithm, we employed a Random Forest (RF) Classifier, which leverages multiple decision trees at training time when predicting the correct label to assign to unseen data. The output prediction of each tree is taken into consideration when calculating the final label to assign via a majority vote. This method, called also bagging or bootstrap aggregation, reduces the variance by decorrelating the output of the different trees by choosing a subset of the predictors considered for each tree [24].

The trained ML models we use for determining the latency classes to feed to the SSA and the IL to feed to the ML-RS algorithm are generated within the 5Gr-MLaaSP using Spark, which leverages a pipeline approach, applying different transformations to the considered dataset (i.e., normalization) combining them into a single workflow. A hyperparameter search is performed on the model in order to obtain the best possible result. The considered hyperparameters are: the maximum depth of the trees, the minimum number of samples required to split an internal node, the minimum number of samples per leaf node, the number of decorrelated trees generated during the training phase, and the split criterion in the trees generation, whether is Gini index or entropy. The values of such hyperparameters, along with the suitable ones, are reported in Tab. II. The test accuracy obtained using the best model hyperparameters for the slice-subnet sharing model is equal to 0.9835, and the related confidence intervals are presented in Tab. III. Similarly, Tab.IV shows the results obtained training the service scaling ML models.

TABLE II: Values of the hyperparameters of the RF model

| **Parameter** | **Tested values** | **Best value** |
|---|---|---|
| Number of estimators | $\{100, 200, 300\}$ | 100 |
| Maximum depth | $[1, 10]$ | 9 |
| Minimum sample per split | $[2, 10]$ | 2 |
| Minimum samples per leaf | $[1, 5]$ | 2 |

TABLE III: RF model for slice-subnet sharing: confidence intervals

| **Confidence level** | **Confidence interval** |
|---|---|
| 90% | [0.98331, 0.98358] |
| 95% | [0.98331, 0.98364] |
| 98% | [0.98332, 0.98365] |
| 99% | [0.98332, 0.98545] |

To find the best combination of hyperparameters, the model has been validated using a *k*-fold cross validation method,

TABLE IV: Accuracy of the service scaling models

| Service | Model accuracy | Confidence interval (95% CL) |
|---|---|---|
| CD | 0.9918 | [0.9914, 0.9921] |
| DaBEV | 0.9912 | [0.9906, 0.9919] |
| ST | 0.9953 | [0.9949, 0.9958] |

which divides the training dataset in $k$ groups and generates $k$ evaluation scores by training the model on $k-1$ folds and testing it on the remaining one. The final result is given by the mean of the calculated scores. We chose $k = 10$ since it provides an estimate with low bias and modest variance.

The training phase, using a 26,000-sample dataset and including the very time-consuming search for the best model hyperparameters configuration, lasted 45 minutes using a machine with a 2.2 GHz Intel i7-8750H processor and a 16 GB DDR4 RAM.

### B. In-testbed validation

We now present the validation of the interaction between the 5Gr-MLaaSP and the 5Gr-entities, as well as of the ML-driven decision-making processes at the 5Gr-VS and the 5Gr-SO.

**Testbed results for slice-subnet sharing at the 5Gr-VS.** The in-testbed validation at the 5Gr-VS demonstrates the feasibility of the proposed architecture and of the ML-based slice-subnet sharing algorithm presented in IV-A. Further, it shows that the impact on the life-cycle management actions, in terms of introduced delays, is negligible. For this, we deployed the 5Growth MANO platform as per the architecture depicted in Fig. 2. This platform is used to manage the collision detection (CD) and see-through (ST) services described in Sec. VI. The 5Gr-VS of the experimental setup implements the architectural blocks and the interaction introduced in Sec. IV-C. It is configured with a default arbitration policy, which determines the trained ML to be retrieved from the 5Gr-MLaaSP, it runs the model to determine the latency classes configuration, and executes the SSA for slice-subnet sharing.

The tests performed to gather the results consisted in repeating ten times the instantiation of the CD and the ST services. During the instantiation of the services at the 5Gr-VS, we measure the statistical distribution of the time required to execute the ML-based model and the SSA. Each run of the tests starts with the instantiation of the CD service. During this operation, the 5Gr-VS maps the service to one NSI containing two NSSIs: NSSI_A implementing the TDSS logic (i.e., TDA and CIM), and NSSI_B implementing the DENM Generator logic of Fig. 6. Upon being executed, the SSA determines that all the NSI and NSSIs of the CD service are to be provisioned since there are no candidate slice subnets to be shared. Thus, the 5Gr-VS provisions the NSI and NSSIs, and requests the corresponding network services to the 5Gr-SO.

Once the CD service has been deployed, the test proceeds with the instantiation of the ST service. In this case the service is mapped to one NSI, and two NSSIs: NSSI_A as before, and NSSI_C containing the AVSS (i.e., Video Server and Video Controller). Given the latency constraints of the new service, in this case the SSA determines that the NSSI_A can be reused and the scaling actions to be performed to accommodate the additional traffic demand.

Fig. 8 presents the results for the CD and ST services. The depicted boxplots cover the experienced maximum, minimum, average, median, 20th and 80th percentile values across the ten performed repetitions. We note that the same boxplot representation is going to be used used in the boxplots graphs presented in the rest of this section. Fig. 8 shows that the delays introduced by the ML-driven latency class configuration and the SSA are approximately equal to 2.5 s for both considered cases. This proves both the feasibility of our approach for ML-driven service provisioning, and the reduced impact in terms of overall service instantiation time.

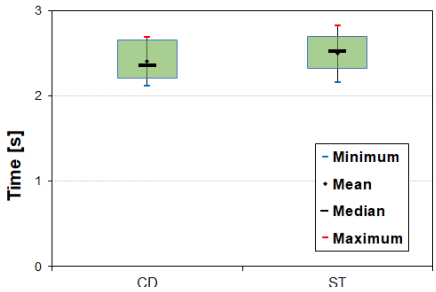



Fig. 8: Time (in seconds) of ML-driven service provisioning for the collision detection and see-through services.

Finally, we remark that, although the results presented in Fig. 8 do not account for the time required to train the ML model within the 5Gr-MLaaSP, our experiments showed that the model training takes about 25 s, which is acceptable, even if the model needed to be retrained with fresher data collected from the monitoring platform at every service instantiation.

**Testbed results for resource scaling at the 5Gr-SO.** As before, we use a 5Growth MANO platform instance as depicted in Fig.2. The platform controls an NFVI infrastructure composed of three NFVI-PoPs, which are managed by dedicated instances of a Virtual Infrastructure Manager (VIM), implemented with Openstack software. A transport network, emulated with GNS3 software, interconnects these NFVI-PoPs. The transport network has five packet-switches following a ring topology of four elements with an additional packet switch in the middle to provide additional path redundancy. These packet switches are controlled by an instance of an ONOS SDN controller. In these experiments, we considered a distributed NFVI-PoP scenario modeling an edge scenario, where resources are limited and may not be enough for the deployment of a whole NSI. Moreover, this kind of distributed deployment allows showing the impact of updating the interconnections between the different NSSIs that are part of an NSI when a scaling operation is performed. It is worth mentioning that this resource-driven handling of the interconnections between NSSIs upon scaling parts of an NSI has been scarcely tackled in the literature (further details can be found in [25]).

Since the focus of the experimental evaluation is on measuring the scaling performance of the system, the deployment of NSIs is as follows. Initially, the 5Gr-VS requests to

the 5Gr-SO the deployment of the CD service as an NSI[5] (NSI_1) is composed of two NSSIs (NSSI_A and NSSI_B), defined as above. Then, the 5Gr-VS requests to the 5Gr-SO the deployment of the ST service (NSI_2) and the 5Gr-VS determines the sharing of the NSSI_A, so that the 5Gr-SO only needs to deploy the remaining AVSS (referred to as NSSI_C) to complete the instantiation of the ST service. In this experiment, each NSSI is deployed in a different NFVI-PoP. During the instantiation of the different NSIs, in particular of NSSI_A and NSSI_C, the 5Gr-SO contacts the 5Gr-MLaaSP to download the required ML model to drive scaling operations. The ML-based scaling decision is driven by the performance of the TDA and Video Server VNF instances present in NSSI_A and NSSI_C, respectively. The following graphs present the experienced time to perform ML-driven scaling operations in the above deployment. Each experiment has been repeated ten times.

Fig. 9 shows the statistical distribution of the time required to scale out NSSI_A and NSSI_C, and the implications of the different performed operations in the overall deployment. As the traffic density increases, the ML model for NSSI_A determines that a new IL including a new instance of the TDA VNF is required to fulfill the service requirements. Consequently, more requests are done to the Video Server VNF, and the ML model for NSSI_C also determines that a new IL including a new instance of this VNF is required.

The scale out of NSSI_A produces changes in the overall deployment affecting also NSSI_B and NSSI_C because both deployed NSIs share the NSSI_A instance. These changes take 47.98 s on average. Out of this time, 86.7% is devoted to adding the new TDA VNF instance. This time also includes the operations carried out by the 5Gr-SO due to the ML-driven scaling procedure followed, namely data-engineering pipeline configurations actions, as explained in the last steps of the workflow presented in Sec. V-C. In total, these operations represent a 5.88% of the average NSSI_A scaling time required to add the new TDA VNF instance. The most time-consuming operation in this set of actions associated with the ML-driven scaling process is the termination of the inference job, running as an Apache Spark job (around 2.5 s on average). This high time is experienced because the inference job is under execution when the termination request arrives prior to proceeding with the scaling operation, as observed in [17]. The rationale of stopping the inference job during the NSSI being scaled is to avoid overloading the 5Gr-SO with wrong scaling decisions issued while NSSI_A is updated.

The remaining 13.3% of the overall deployment scaling time is devoted to connecting this new TDA VNF instance with the VNFs deployed to associated NSSIs, namely, NSSI_B and NSSI_C. The average time required to update the interconnections with NSSI_C is larger (3.431 s vs. 2.960 s) than the one with NSSI_B because NSSI_C includes two VNFs.

When scaling out the Video Server VNF of the NSSI_C, the overall deployment scaling time (37.94 s on average) is lower due to two facts. On one hand, the scaling of NSSI_C only requires the update of the interconnections with a single NSSI, i.e., the associated NSSI_A. On the other hand, there is a difference in performance in the hardware running in the NFVI-PoPs where NSSI_A and NSSI_C are deployed. In these experiments, all VNF descriptors considered the same characteristics in terms of resources (i.e., CPU, RAM, and storage). While it takes 41.59 s on average to scale just the NSSI_A (i.e., add the new instance of the TDA VNF), the same operation for NSSI_C (i.e., adding a new VNF instance) takes 34.14 s on average. In this case, the impact of ML-related operations follows the same trends as before and represents the 7.19% of the NSSI_C scaling time. This is a higher percentage than before because the overall deployment scaling time is lower for this case, but the time required by the ML operations is approximately the same in both cases.

Fig. 10 shows the statistical distribution of the experienced time when scale-in operations occur, e.g., due to a decrease in the traffic density, the ML model determines that NSSI_C and NSSI_A can return to their initial IL (24.91 s and 34.48 s, respectively). The overall observed trends are the same as with the scale-out operations explained before. Regarding the impact of ML operations (10.22% for NSSI_C and 7.33% for NSSI_A), the most time-consuming operations is the termination of the inference job before the deletion of the corresponding VNFs required by the new target IL. In this case, the experienced scaling time is lower mainly because the time to deallocate resources (VMs associated with the VNFs and transport network connectivity services) is smaller than to allocate them. We also observe the difference in performance between the hardware of the different NFVI-PoPs hosting the VNFs of the different NSSIs.

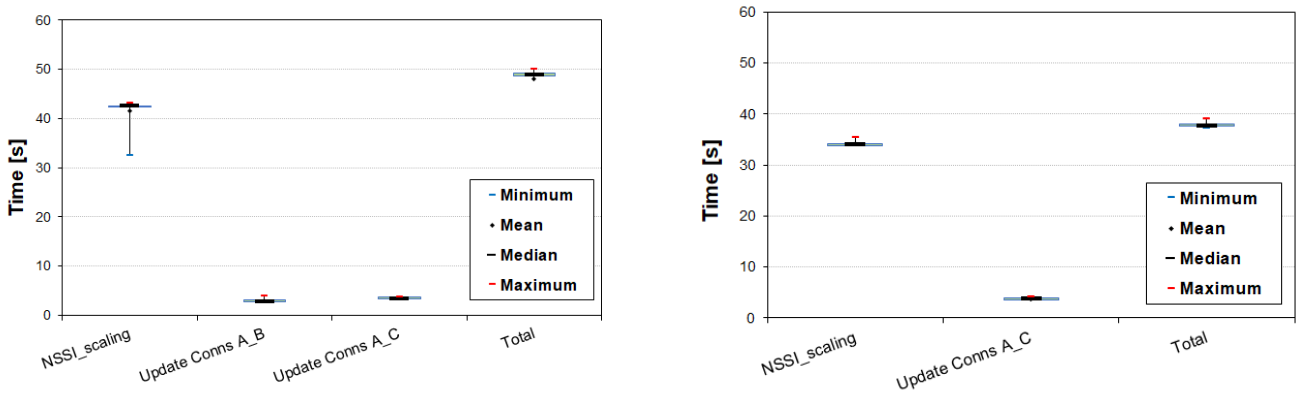
In summary, for the deployed services under evaluation, ML-related operations have taken, on average, roughly from 5% to 10% of the total scaling time. Since these operations take approximately the same time independently from the service, the higher shares are obtained when scaling operations take less time in absolute terms (e.g., allocation/deallocation of VMs for scale out/in operations or less inter-nested or inter-PoP connections to be established).

### C. Numerical results in a large-scale scenario

We now consider the real-world, large-scale scenario described in Sec. VII-A and assess the performance of the proposed solutions via simulation.

**Simulation results for slice-subnet sharing at the 5Gr-VS.** Fig. 11(left) shows the performance gain in percentage with respect to the case where no NSSI sharing is applied. Specifically, the results are presented in terms of savings in number of vCPU utilization and in number of VMs instantiated, and for different values of vehicle arrival rate. As expected, the gain decreases as the workload grows, however it is interesting to notice that NSSI sharing allows for substantial saving of computing resources. The fluctuation of the number of active VMs is due to the fact that the best latency class configuration selected through the ML approach aims at the minimization of the vCPU consumption, which may not be the same as the configuration that would minimize the number of used VMs.

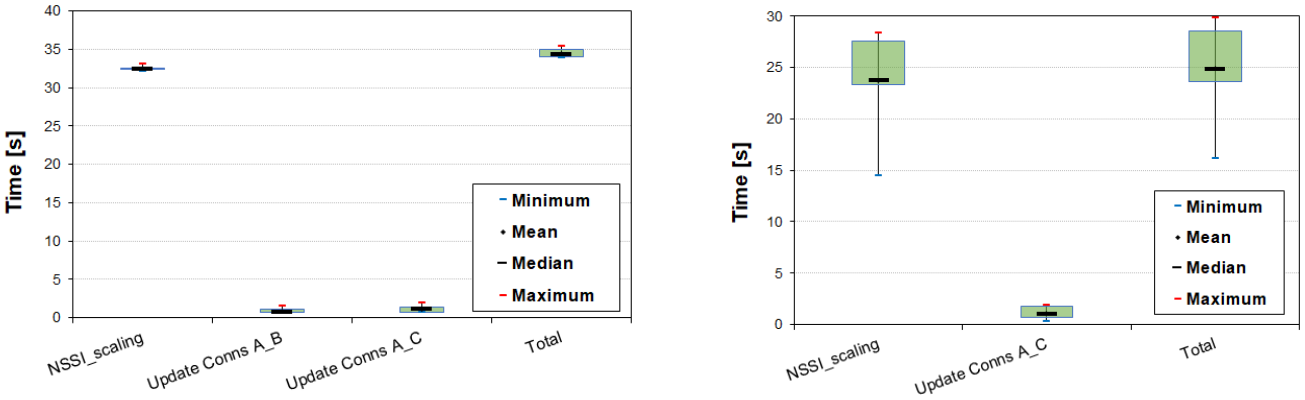
[5]As mentioned in Sec. III, the NSIs are requested by the 5Gr-VS to the 5Gr-SO in the form of composite NFV-NSs, and its nested NFV-NSs become the NSSIs of the service.




(a) NSSI_A scale out operation

(b) NSSI_C scale out operation

Fig. 9: ML-driven scale-out of an NSI shared deployment.




(a) NSSI_A scale in operation

(b) NSSI_C scale in operation

Fig. 10: ML-driven scale-in of an NSI shared deployment.

Under the same scenario, Fig. 11(middle) shows the average number of instances that share the same NSSI. Interestingly, we notice that, as the vehicle arrival rate increases (hence the number of ST and DaBEV requests grows), the number of instances sharing the same Trajectory Detection slice subnet decreases more slowly than in the case of the Adaptive Video slice subnet. Indeed, ST and CD have a much more similar target latency than ST and DaBEV, thus making the benefit of the latter two services sharing a NSSI drop faster.

Next, in Fig. 11(right) we investigate the impact of the percentage of ST and DaBEV requests for different values of vehicle arrival rates. The plot confirms that the vCPU gain with respect to the case of no NSSI sharing is substantial for lower rates and vanishes in the highest rate scenario, as the best latency class configuration tends to place all the NSSIs in different latency classes.

In conclusion, the above results demonstrate that the proposed ML-drive slice-subnet sharing for service provisioning is highly beneficial, with up to 40% reduction of vCPU in light load conditions, and a reduction of the number of instantiated VMs that ranges between 20% and 40%.

**Simulation results for resource scaling at the 5Gr-SO.** We now evaluate the performance of the SLA management solution proposed for OPEX minimization. Three different strategies are compared: the L-INS and H-INS strategies, where NFV-NSs are instantiated using the "small IL" and "big IL" instantiation levels (resp.) and such ILs are never changed, and our proposed ML-RS solution. Note that, in this case, "small IL" and "big IL" correspond to 2 and 5 (resp.) instances of the bottleneck VNFs of the NFV-NSs. Moreover, according to the target values in Sec. VI and our experimental implementation of the services, we set the latency thresholds, $\delta$, for the nested NFV-NSs to: 5 ms, for the TDSS, 120 ms for the ST AVSS, and 950 ms for the DaBEV AVSS. The SLA violation penalty associated with the latency thresholds is $p = 2$ unit per second. Moreover, we set $\sigma_{\mathsf{ins}} = 1,000$ units, and $\sigma_{\mathsf{opr}} = 0.12$ unit/s.

To evaluate the efficiency of the strategies, their performance as a function of traffic load is compared in Fig. 12 where the SLA violation cost, the service provisioning cost, which is the sum of instantiation and operation costs, and the total OPEX cost per strategy are depicted. These results, obtained by averaging over 20 runs, are presented as functions of parameter $\ell$, by which the arrival request rate is multiplied. Also, the costs are computed over a 24-hour period in Fig. 12(a)-(b), and over a 48-hour period in Fig. 12-(c).

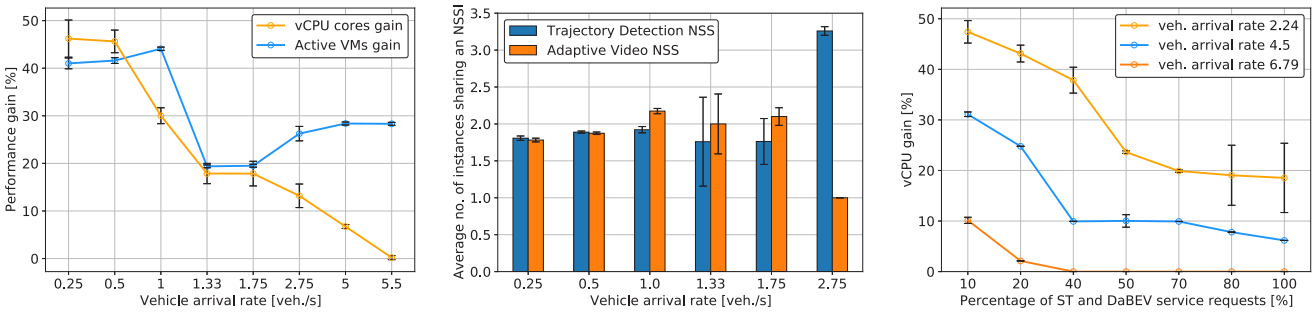



Fig. 11: Simulation results on slice-subnet sharing vs. vehicle arrival rate. Left: gain in terms of vCPU consumption and number of active VMs, compared to no sharing (ST and DaBEV set to 50% and 30%, resp.); Middle: average no. of instances sharing the same NSSI (ST and DaBEV set to 50% and 30%, resp.); Right: vCPU consumption gain vs. percentage of ST and DaBEV requests, relatively to no sharing and for different vehicle arrival rates. Each plot reports also the 95% confidence interval over 100 simulations (in some cases it is so small to be scarcely visible).

We observe that, under lightly loaded conditions, i.e., $\ell \leq 0.3$, all strategies comply with the SLA, so there is no SLA violation cost. However, because of over-provisioning, the OPEX of H-INS is significant while L-INS and ML-RS use the minimum number of instances and have the same OPEX. By increasing $\ell$, the SLA violation cost of L-INS grows exponentially but the proposed solution can maintain a negligible SLA violation by efficient resource scaling that minimizes the OPEX. More specifically, when $0.3 < \ell < 1.0$, ML-RS yields the same SLA violation cost as the H-INS, but with a considerable lower OPEX due to the lower provisioning cost via the appropriate resource scaling. In the highly loaded conditions where $\ell > 1$, even the "big IL" is not sufficient to handle the offered load, i.e., the H-INS-SLA $> 0$. In this case, ML-RS at the beginning scales out the service and never scales it in, which makes H-INS and our solution have similar performance.

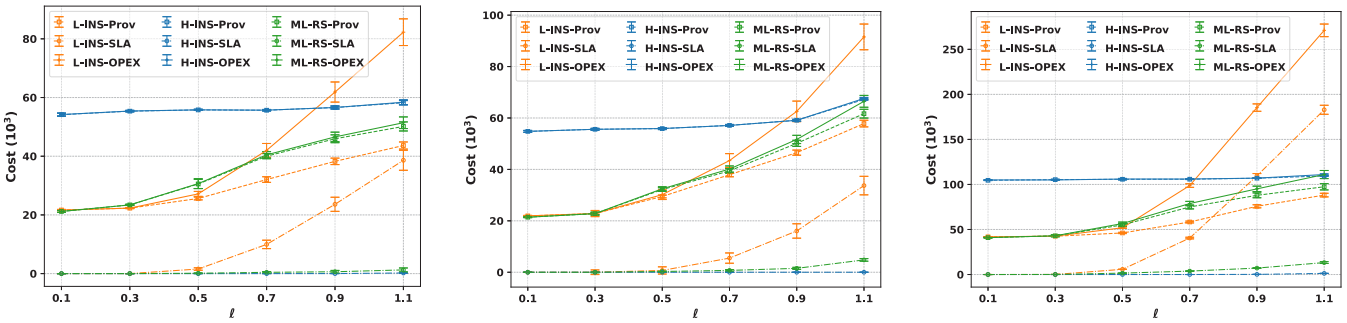
In summary, these results show how the ML-based approach is capable of adapting the service setup (in this case, IL) to improve the OPEX for the 5Growth provider by finding an appropriate trade-off between SLA compliance and service provisioning cost. Remarkably, the OPEX is halved in comparison to the overprovisioning strategy in lightly loaded conditions, while the SLA violation is negligible in highly loaded conditions.

## VIII. Related Work

In new-generation cellular networks, services are provided through the provisioning of logical networks according to the well-known network slicing paradigm, on which useful surveys can be found in [26], [27], [28], [29], [30]. In this context, the MANO architecture [9] is often used for the life cycle management and the runtime operation of network slices. Beside the 5Growth project [2], also 5GCity [31] proposes an orchestration platform, while 5G-MoNArch [32] aims at developing an experiential network intelligence that combines AI/ML with network orchestration and management. Additionally, there are similarities between our proposed architecture and that of the MATILDA project [33]. The emphasis of this paper, however, is on the realization of the MLaaS concept through the creation of a separate building block (5Gr-MLaaSP) providing an ML catalog that can be consumed by any other block of the architecture. In this sense, the 5Gr-MLaaSP can serve the needs of a wide variety of current problems related to service management, and those that will appear. On the other hand, learning processes in the MATILDA architecture are bound to the intelligent service orchestrator, and, hence, are integrated with the logic of this block, which focuses on specific problems. We believe that an external 5Gr-MLaaSP offers a generic and future-proof architectural solution, which also follows the recommendations of such SDOs as 3GPP or O-RAN [4]. It is also worth mentioning that, as in the case of the 5Growth architecture, the MATILDA project [34], [35] envisions a separation of concerns between the vertical application domain and the network operator domain where slices are deployed.

ML techniques are also leveraged in [36] for radio resource scheduling and management, and in [37] for resource orchestration and an optimal usage of physical resources. In [38], deep reinforcement learning is investigated for network slice reconfiguration, with the aim to minimize long-term resource consumption. Particularly relevant to our work are also the studies in [39], [40]. Indeed, [39] proposes a hierarchical orchestration architecture to deal with multi-domain scenarios, as well as a service auto-scaling algorithm. The latter foresees both a ML-driven proactive provisioning technique and a reactive resource adaptation, so that the service target latency can be met in spite of the time-varying traffic demand. [40], instead, focuses on a negotiation game between verticals and network providers, for service chains auto-scaling. Interestingly, [40] proposes an ML-driven scaling decision process at the service orchestrator, which however does not account for the trade-off between the cost of resources and that of SLA violations and is evaluated through numerical tests only. Such a trade-off is instead investigated in [41], where an ML-based scaling management, specifically designed for Kubernetes edge clusters, is presented.

Several works have adopted algorithmic approaches to resource allocation and service admission control. Among these studies, [42] proposes elastic NFV resource allocation according to predefined resource pressure thresholds, while




(a) Collision Detection service in a 24 hour period (b) Bird Eye service in a 24 hour period (c) Bird Eye service in a 48 hour period

Fig. 12: Provisioning cost, SLA violation cost, and OPEX with respect to the scale of the arrival rate of the service requests. Each plot reports also the 95% confidence interval over 20 simulations.

[43] predicts scaling events feeding application-level metrics to a neural network, which however poses privacy concerns. Also, [44] leverages deep learning models to reliably and dynamically orchestrate end-to-end slices, and allocate radio, computing, and storage resources. Others focus on admission control aspects, exploiting ML techniques [45] or auction mechanisms and game theory [46], [47].

It is worth underlying that all the above studies have a different scope from that of slice-subnet sharing that we consider. Indeed, they focus on resource allocation or sharing among different network slices, instead of network slice-subnet sharing among different services. Further, our work provides a complete, thorough evaluation of the proposed solutions, via both large-scale simulations and experimental tests carried out through our testbed. Importantly, the latter ones also demonstrate the interaction and functional synergy between the 5G architecture entities, thus validating our system design. Finally, we mention that a preliminary version of the ML-driven service scaling presented in this work has been presented and demonstrated in our conference [17] and demo [48] papers.

## IX. Conclusions

We tackled the problem of fully-automated service provisioning and management in a virtualized 5G network platform. We proposed a system design that allows the 5Gr-VS and 5Gr-SO architectural layers to effectively interact with the MLaaS platform we created, and to leverage ML-driven decision-making processes. We also defined algorithmic solutions for ML-driven slice-subnet sharing and run-time service scaling, under dynamic traffic conditions. Through in-testbed validation, we demonstrated the feasibility of our approach, the designed functional synergy between the 5G architecture entities, as well as the limited time overhead introduced by the ML framework. Further, simulation results in a large-scale, real-world scenario showed remarkable savings in operational costs, e.g., up to 40% reduction in vCPU consumption and up to 30% reduction in the OPEX.

## References


[1] "5G-CLARITY project: Initial design of the SDN/NFV platform and identification of target 5G-CLARITY ML algorithms," *D4.1*, 2020.

[2] "5GROWTH: 5G-enabled growth in vertical industries," https://5growth.eu/journals-and-magazines, [Accessed in December 2021].

[3] ETSI, "Zero-touch network and Service Management (ZSM); Reference Architecture," Tech. Rep., 2019.

[4] 3GPP, "O-RAN Working Group 2: AI/ML workflow description and requirements," Tech. Rep. O-RAN.WG2.AIML, 2020.

[5] ETSI, "GR NGP 011: Next Generation Protocols (NGP); E2E Network Slicing Reference Framework and Information Model," 2018.

[6] 3GPP, "3GPP TS 28.530 v17.11.0 - 5G; Management and orchestration; Concepts, use cases and requirements (Release 17)," 2021.

[7] X. Li, F. Chiasserini, C. J. Mangues-Bafalluy, J. Baranda, G. Landi, B. Martini, X. Costa-Perez, P. Puligheddu, and L. Valcarenghi, "Automated service provisioning and hierarchical SLA management in 5G systems," *IEEE Trans. on Network and System Management*, vol. 18, no. 4, pp. 4669–4684, 2021.

[8] 3GPP, "3GPP TS 28.533 v16.7.0 - Technical Specification Group Services and System Aspects; Management and orchestration; Architecture framework (Release 16)," 2021.

[9] ETSI, "NFV Release 2 Description," https://docbox.etsi.org/ISG/NFV/Open/Other/ReleaseDocumentation/NFV(21)000023_NFV_Release_2_Description_v1_12_0.pdf, [Accessed in December 2021].

[10] 3GPP, "TS 28.536 v16.3.0 - Technical Specification Group Services and System Aspects; Management and orchestration; Management services for communication service assurance (Release 16)," 2021.

[11] ——, "TS 23.288 v17.0.0 - Technical Specification Group Services and System Aspects; Architecture enhancements for 5G System (5GS) to support network data analytics services (Release 17)," 2021.

[12] M. J. Neely, E. Modiano, and C.-P. Li, "Fairness and optimal stochastic control for heterogeneous networks," *IEEE/ACM Trans. on Networking*, vol. 16, no. 2, pp. 396–409, 2008.

[13] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *IEEE CLOUD*, 2010, pp. 370–377.

[14] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "VNF placement and resource allocation for the support of vertical services in 5G networks," *IEEE/ACM Trans. on Networking*, vol. 27, no. 1, pp. 433–446, 2019.

[15] J. Prados, P. Ameigeiras, J. J. Ramos-Munoz, J. Navarro-Ortiz, P. Andres-Maldonado, and J. M. Lopez-Soler, "Performance modeling of softwarized network services based on queuing theory with experimental validation," *IEEE Trans. on Mobile Computing*, vol. 20, no. 4, pp. 1558–1573, 2021.

[16] J. Prados-Garzon, J. J. Ramos-Munoz, P. Ameigeiras, P. Andres-Maldonado, and J. M. Lopez-Soler, "Modeling and dimensioning of a virtualized MME for 5G mobile networks," *IEEE Trans. on Vehicular Technology*, vol. 66, no. 5, pp. 4383–4395, 2017.

[17] J. Baranda and et al., "On the integration of AI/ML-based scaling operations in the 5Growth platform," in *IEEE NFV-SDN*, 2020, pp. 105–109.

[18] 5G-Transformer, "D1.1, Report on Vertical Requirements and Use Cases," Technical Report, 2018.

[19] M. Malinverno, G. Avino, C. Casetti, C. F. Chiasserini, F. Malandrino, and S. Scarpina, "Edge-based collision avoidance for vehicles and vulnerable users: An architecture based on MECs," *IEEE Vehicular Technology Mag.*, vol. 15, no. 1, pp. 27 –35, 2019.

[20] M. Malinverno, J. Mangues, C. Casetti, C. Chiasserini, M. Requena, and J. Baranda, "An Edge-based Framework for Enhanced Road Safety of

Connected Cars," *IEEE Access, March 2020*, vol. 8, pp. 58 018–58 031, Mar. 2020.
[21] G. Avino and et al., "A MEC-based extended virtual sensing for automotive services," *IEEE Trans. on Network and Service Management*, vol. 16, no. 4, pp. 1450–1463, 2019.
[22] F. Rameau, H. Ha, K. Joo, J. Choi, K. Park, and I. S. Kweon, "A real-time augmented reality system to see-through cars," *IEEE Trans. on Visualization and Computer Graphics*, vol. 22, no. 11, pp. 2395–2404, 2016.
[23] M. Rapelli, C. Casetti, and G. Gagliardi, "Vehicular traffic simulation in the city of turin from raw data," *IEEE Trans. on Mobile Computing*, pp. 1–1, 2021.
[24] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2nd ed. Springer, 2009.
[25] 5GROWTH, "D2.3, Final Design and Evaluation of the innovations of the 5G End-to-End Service Platform," Technical Report, 2021.
[26] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.
[27] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hinesd, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, February 2020.
[28] F. Debbabi, R. Jmal, L. C. Fourati, and A. Ksentini, "Algorithmics and modeling aspects of network slicing in 5G and beyonds network: Survey," *IEEE Access*, vol. 8, pp. 162 748–162 762, 2020.
[29] M. Chahbar, G. Diaz, A. Dandoush, C. Cérin, and K. Ghoumid, "A comprehensive survey on the E2E 5G network slicing model," *IEEE Trans. on Network and Service Management*, vol. 18, no. 1, pp. 49–62, 2021.
[30] J. Gil Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.
[31] H. Khalili and et al., "Network slicing-aware NFV orchestration for 5G service platforms," in *EuCNC*, 2019, pp. 25–30.
[32] D. M. Gutierrez-Estevez, N. Dipietro, A. Dedomenico, M. Gramaglia, U. Elzur, and Y. Wang, "5G-MoNArch use case for ETSI ENI: elastic resource management and orchestration," in *IEEE CSCN*, 2018, pp. 1–5.
[33] "MATILDA project: Intelligent orchestration mechanisms," *D3.2*, 2020.
[34] "MATILDA project: 5G-ready vertical applications orchestration," *White paper*, Dec. 2019.
[35] R. Bruschi, F. Davoli, C. Lombardo, and J. F. Pajo, "Managing 5G network slicing and edge computing with the MATILDA telecom layer platform," *Computer Networks*, vol. 194, July 2021.
[36] D. M. Gutierrez-Estevez and et al., "Artificial intelligence for elastic management and orchestration of 5G networks," *IEEE Wireless Communications*, vol. 26, no. 5, pp. 134–141, 2019.
[37] Q. Liu and T. Han, "VirtualEdge: Multi-domain resource orchestration and virtualization in cellular edge computing," in *IEEE ICDCS*, 2019, pp. 1051–1060.
[38] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y.-C. Liang, "Network slice reconfiguration by exploiting deep reinforcement learning with large action space," *IEEE Trans. on Network and Service Management*, vol. 17, no. 4, pp. 2197–2211, 2020.
[39] I. Afolabi, J. Prados-Garzon, M. Bagaa, T. Taleb, and P. Ameigeiras, "Dynamic resource provisioning of a scalable E2E network slicing orchestration system," *IEEE Trans. on Mobile Computing*, vol. 19, no. 11, pp. 2594–2608, 2020.
[40] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling network service chains using machine learning and negotiation game," *IEEE Trans. on Network and Service Management*, vol. 17, no. 3, pp. 1322–1336, 2020.
[41] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine learning-based scaling management for Kubernetes edge clusters," *IEEE Trans. on Network and Service Management*, vol. 18, no. 1, pp. 958–972, 2021.
[42] H. Yu, J. Yang, and C. Fung, "Fine-grained cloud resource provisioning for virtual network function," *IEEE Trans. on Network and Service Management*, vol. 17, no. 3, pp. 1363–1376, 2020.
[43] L. Cao, P. Sharma, S. Fahmy, and V. Saxena, "ENVI: Elastic resource flexing for network function virtualization," in *USENIX HotCloud*, Santa Clara, CA, Jul. 2017.
[44] H. Chergui and C. Verikoukis, "Offline SLA-constrained deep learning for 5G networks reliable and dynamic end-to-end slicing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 350–360, 2020.
[45] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing," in *IEEE INFOCOM*, 2017, pp. 1–9.
[46] B. Han, V. Sciancalepore, X. Costa-Pérez, D. Feng, and H. D. Schotten, "Multiservice-based network slicing orchestration with impatient tenants," *IEEE Trans. on Wireless Communications*, vol. 19, no. 7, pp. 5010–5024, 2020.
[47] T. V. K. Buyakar, H. Agarwal, B. R. Tamma, and A. A. Franklin, "Resource allocation with admission control for GBR and delay QoS in 5G network slices," in *COMSNETS*, 2020, pp. 213–220.
[48] J. Baranda, J. Mangues-Bafalluy, E. Zeydan, C. Casetti, C. F. Chiasserini, M. Malinverno, C. Puligheddu, M. Groshev, C. Guimaraes, K. Tomakh, and O. Kolodiazhnyi, "Demo: AIML-as-a-Service for SLA management of a Digital Twin virtual network service," in *IEEE INFOCOM - Demo Session*, 2021.

**Claudio Casetti** (SM'14) worked as a visiting researcher at UCLA and UCSD, and as a Visiting Professor at Monash University. He is currently a Professor at Politecnico di Torino.

**Carla Fabiana Chiasserini** (F'18) worked as a visiting researcher at UCSD and as a Visiting Professor at Monash University in 2012 and 2016. She is currently a Professor at Politecnico di Torino and EiC of Computer Communications.

**Silvio Marcato** received his MSc degree in 2021 from Politecnico di Torino, where he is currently a Research Fellow.

**Corrado Puligheddu** (M'20) received his MSc degree in 2019 from Politecnico di Torino, where he is currently a PhD student. His research interests are in wireless networks and mobile applications.

**Josep Mangues-Bafalluy** (Ph.D. 2003 UPC) is Research director and Head of the Services as networkS (SaS) research unit of the CTTC. He has participated in and led numerous research projects on autonomous network management.

**Jorge Baranda** (Ph.D 2011, SM'19) is a Researcher in the Services as networkS (SaS) research unit at CTTC. He has participated in several national and international projects, on SDN/NFV based orchestration of mobile networks.

**Juan Brenes** (M.Sc. 2015) is a network architecture researcher and developer at Nextworks.

**Francesco Bocchi** (B.Sc. 2020) currently works at Nextworks, his interests include NFV MANO orchestration technologies.

**Giada Landi** (M.Sc. 2005) is R&D leader of architectures and network design at Nextworks. She has participated in many industrial and European research projects; here interests are in control plane architectures and protocols, SDN and NFV.

**Bahador Bakhshi** (Ph.D. 2011) is a Researcher in the Services as networkS (SaS) research unit of the CTTC. In 2012-2020, he worked as assistant professor with the Computer Engineering Department of Amirkabir University of Technology.