Dynamic Slicing Reconfiguration for Virtualized 5G Networks Using ML Forecasting of Computing Capacity

Juan Sebastian Camargo*[1], Estefanía Coronado[1,6], Wilson Ramirez[1], Daniel Camps[1], Sergi Sánchez Deutsch[1], Jordi Pérez-Romero[2], Angelos Antonopoulos[3], Oscar Trullols-Cruces[3], Sergio Gonzalez-Diaz[4], Borja Otura[4], and Giovanni Rigazzi[5]

[1] i2CAT Foundation(juan.camargo, estefania.coronado, wilson.ramirez, daniel.camps, sergi.sanchez)@i2cat.net
[2] Dept. Signal Theory and Communications Universitat Politecnica de Catalunya (UPC)(jordi.perez-romero@upc.edu)
[3] Nearby Computing S.L.(aantonopoulos, otrullols)@nearbycomputing.com
[4] Atos(sergio.gonzalez.diaz, borja.otura)@atos.net
[5] Cellnex Telecom(giovanni.rigazzi@cellnextelecom.com)
[6] High-Performance Networks and Architectures, Universidad de Castilla-La Mancha(estefania.coronado@uclm.es)

Abstract. As 5G deployments continue to increase worldwide, new applications can fully leverage the exceptional features of the emerging mobile networks. Ultra-Reliable Low Latency Communications (URLLC) serve as an excellent example of applications highly sensitive to jitter and packet loss. To meet these demanding requirements, 5G relies on network slicing, network virtualization, and software-defined networks. This ecosystem enables the precise allocation of resources for each network slice. However, the applications' resource demands may vary over time. In this challenging and overwhelming environment, traditional human decision-making for slice reconfiguration is not suitable anymore, due to the multitude of parameters and the need for extremely fast response times. Machine Learning (ML) comes as a tool that can enable better use of the available resources with faster and more intelligent management. This paper introduces an ML model that can predict slices' traffic and dynamically reconfigure computational capacity. With these forecasting capabilities, the virtualized resources can be fine-tuned to suit the slices' requirements, guaranteeing their Quality of Service (QoS). By doing so, Mobile Network Operators can make optimized use of the equipment, tailoring their needs to each service while complying with the QoS level. The results obtained demonstrate that the proposed ML model, in combination with a specific set of hysteresis rules, can accurately predict the saturation of virtualized capacity with up to 91% accuracy and proactively adapt it to the network slice requirements.

Keywords: O-RAN · NFV · Resource Forecasting · AI · ML · Network Reconfiguration · Kubernetes

*Corresponding author.

1 Introduction

Mobile Network Operators (MNOs) are trying to fulfill the ever-increasing network technical requirements while adapting to service flexibility and reducing the operational expenditures in their networks [1]. For that reason, as MNOs look to reduce the cost, they also need to fulfill high service level agreements from a wide variety of services and applications running in their networks. Under this scenario, MNOs are currently facing an opposing problem: they need to provide highly differentiated and demanding services while at the same time being limited in terms of network capacity. Therefore, to cope with such an extreme scenario, Open RAN 5G architectures follow the principle of complementing 3GPP standards with virtualized elements, as described in [2].

While virtualization, orchestration, and network slicing appear as big opportunities to overcome these issues, they are also challenging technologies to adopt [3]. However, virtualization generates an under/over provisioning dilemma in the MNOs, as the allocated resources by the orchestration should be enough to satisfy the Service Level Agreement (SLA) of a variety of applications, but at the same time those resources should be the minimum possible, to save unwanted operational and capital costs. One solution to balance these needs is to be able to predict in advance the incoming traffic that services and applications will handle, and then alert the network administrator to adjust the virtualized capacity accordingly. Nevertheless, this approach still requires human intervention, whose reaction time would be prone to high delays in the network's reconfiguration. In this regard, Machine Learning (ML) approaches have been proven to serve the purpose of dynamism required in virtualized resource provisioning problems in various orchestration and networking platforms [4, 5]. However, although several works in this area have attempted to solve this problem, the number of incorrect predictions due to too frequent and drastic changes in the resource demands, create unnecessary operational costs and SLA breaches. Moreover, it must be taken into consideration that most of the slicing reconfiguration management systems proposed in the literature are evaluated under simulation or lab environment testbeds, which cannot guarantee that the proposed solutions would fit the requirements of real-world commercial 5G networks following internationally accepted standards and equipment.

With that objective in sight, in this paper, we propose a network orchestration system that leverages an ML model to predict the computing resources needed to be allocated to a specific network slice deployed under virtualized equipment. In this way, the ML model allows the network orchestrator to dynamically adapt the slice computing capacity whenever the prediction crosses a pre-defined threshold and the current virtualized resources are not enough to maintain the Quality of Service (QoS) expected to be maintained for the slices. Therefore, this paper pursues two main objectives: (i) to configure the requirements of each application while guaranteeing the agreed Quality of Ser-

vice under such slice, avoiding hysteresis in the resource allocation operations across time due to drastic and fast changes (typical from mobile networks); and (ii) to use only the resources needed for each case, optimizing the network's capacity, and avoiding unnecessary expenses due to SLA breaches or resource over-provisioning. This approach allows MNOs to predict the incoming CPU capacity and adjust it by setting different values to different slice types, optimizing the resource utilization.

The rest of this paper is organized as follows. Section 2 describes the related work in the field of ML-aided orchestration, the main gaps in the existing studies and the novelty of our work. Section 3 introduces the 5G architecture and system model that our work takes a reference. Section 4 describes the proposed ML model and its introduction in the networked system. Moreover, Section 5 discusses the test-bed used for evaluation and the obtained results. Finally, Section 6 draws the conclusions.

2 Related Work

As thoroughly described in [6], the new virtual capacities of a 5G network have led to a position in which the number of configurable parameters and configuration variables is too complex for manual intervention. Additionally, the network equipment capacity has been controlled and virtualized using Network Functions Virtualization (NFV) and Software Defined Networking (SDN) respectively, generating a highly flexible management of the network. However, with virtualization also comes the under/over provisioning of resources compared with the real capacity needed on each piece of equipment. Consequently, in the literature, there can be found several works that aim to tackle different angles of this problem.

On the one hand, the work in [4] describes an ML-based network resource orchestration model that is in charge of predicting the incoming traffic per slice and then assigning the amount of resources needed accordingly. The ML-model is based on a three-dimensional Convolutional Neural Network (CNN) that uses time as an input, allowing it to integrate a temporal variable that provides the trend and proneness of cyclical use of the network into the learning mechanism.

On a similar line, the authors of [5] take the over-provisioning problem of Docker-based apps deployed in Kubernetes and propose an ML model that allows the detection of overbooking resources. Using the variables at the disposal by the containerization tool, the authors use different classifier ML models that allow them to find which of the different variables used as input are part of an overbooked system.

Following this ML trend, the authors of [7] use two types of Long Short Term Memory (LSTM) to predict the performance of the load of Virtual Network Functions (VNF) in the pathway of end-to-end services. The authors compare the performance of a traditional LSTM model against the Context and Aspect embedded attentive Target LSTM (CAT LSTM). The classical LSTM uses only the historical data of each VNF, whereas CAT LSTM uses the same historical data plus the contextual historical data of the neighbors' VNF. The basis of

neighbors' data is also employed by the authors of [8], who introduce a Graph Neural Network (GNN) to predict the future resource requirements of each VNF. The model uses a set of network parameters (CPU, RAM, latency and call drop rate) and as part of the pre-processing data, and generates a profile of the parameter utilization. The model uses the profile of a given VNF component and also the neighbors profile to predict the resource consumption of said VNF component.

In the same trend, the work in [9] also predicts the VNF resource demands using two LSTM models: the previously described CAT LSTM and the Target-Dependent LSTM (TD LSTM). The authors also use a pre-processing method to classify what data is the most important in the forecasting stage, helping the predictors by only using the data that provides good results. Additionally, contrary to the previously described methods, the input of the ML model is not only a quantitative network parameter, but it is a combination of numerical network information with a qualitative status of each of the VNF's studied.

Along with the previous works, the paper in [10] uses an ML model embedded in the orchestrator that works with a Reinforcement-Learning (RL) system. This model takes scenarios and selects actions based on a reward system. The main objective of the model is to observe the current network status and suggest the best-use policy for the VNF associated. The RL agent analyzes different values of the network, including the operational costs, bandwidth availability and virtual links along with the new IoT service instantiation request. With that information, the model can provide suggestions for creating or modifying the existing VNF's in a priority-wise approach.

In the same line, the authors of [11] propose also an RL model using a Deep Neural Network that helps the model to perform adaptive provisioning of VNF at the edge of 5G networks, while looking to maximize the throughput of the VNF. The model takes as input the information of the nodes and the edge infrastructure, along with the delay requirements of the applications. Additionally, the work also highlights the flexibility capabilities of the ML models, as the tests take into consideration training the model for a specific VNF and Edge network and, without re-train the model, the VNF and the Edge size are changed.

Following the Kubernetes approach, the authors of [12] take advantage of the already in-built auto-scaling feature in the containerization platform. By using a winner-takes-all approach, the authors create a competition of different ML models and then, the model that better suits the current situation is chosen as the final predictor. The authors also show that with this forecasting information, it is possible to improve the built-in auto-scaling feature and significantly reduce resource provisioning issues.

It is worth noticing that the works in [5] and in [12] do not evaluate the impact of network traffic on the overall resource utilization of a 5G network. This is an important difference with respect to the work presented in this paper, which uses forecasting techniques to detect over utilization of resources under different traffic schemes in a 5G virtualized network.

Similar to the work presented in this paper, the authors in [13] also use an ML for predicting both resource usage and performance of a virtualized Fog-to-Cloud Environment. However, their approach is different from this study because: 1) a virtualized 5G access network aligned with the O-RAN vision is not used; 2) distinct network traffic schemes were not considered.

Centering on the O-RAN environment, the authors of [14] present an exhaustive description of the current cellular technologies and compare and contrast them with the incoming of O-RAN architecture. The authors also show the evolution of mobile networks and present the current challenges and the future opportunities to adopt the O-RAN architecture. About the current O-RAN architecture, the study presented in [15] provides a comprehensive overview of the O-RAN solution, giving a detailed description of the architecture, the various interfaces, and security considerations at each stage of the architecture. The authors' study highlights how the O-RAN architecture and interfaces help in integrating Artificial Intelligence (AI) and ML techniques creating a new intelligent RAN. Following with ML line, the authors of [16] face the challenges and potential risks of the ML life-cycle while implementing AI in an O-RAN architecture. The authors use as a base the already existing principles for ML Ops and tweak them to the new O-RAN environment.

Regarding slicing, the authors of [17] propose a case of network slicing in an O-RAN architecture. The slicing approach is performed solely in the RAN domain, looking to optimize the physical resources and power allocation of the radio units. The optimization problem is then solved using a mixed-integer non-linear programming and then validated using a simulation. It is worth noting that using a traditional optimization solving method could potentially not be suitable for a real-life implementation, due to the lack of flexibility of the methods, the high level of fine-tuning of the models, and the long time spent to provide decisions that are required to be made in real-time. By using ML, this problem is overcome thanks to the high level of adaptability, flexibility and a more suitable time frame to provide decisions of ML models, as used in our approach.

Another work using ML in an O-RAN architecture is presented in [18], where the authors use an ML model to perform an early attack detection working alongside the O-RAN intelligent controller. In this case, the authors focus only on the classification of the incoming traffic through the air interfaces and, taking advantage of the modularity of the O-RAN architecture, stop the attack before it reaches higher instances of the network. Slicing is, however, out of the scope of this work.

Having considered different solutions for the adaptive configuration of resources in virtualized environments, the work presented in this paper shows a key advantage: it has been specifically designed to meet the elasticity requirements of network slices (allowing to meet strict SLAs while minimizing the resource usage and the investment in equipment), and it has been tested under a real implementation of virtualized 5G networks, following internationally accepted standards and in line with the current state of the art of 5G architecture. By implementing the model proposed in a real deployment site, we guarantee that

no bias or gap could have arisen from testing it in a simulation platform or other type of specifically designed testbeds. Additionally, several studied methods have included the historical information of the predictors to improve their forecasting accuracy. However, forecasting is not 100% accurate and in some cases, it might generate false positives, instances where an increase in forecasting is predicted but in reality, the traffic does not increase. These false positives could generate a bad impact on the performance of the network as actions would be taken, but variations in the traffic managed would not fit said actions, therefore creating additional unwanted operational costs. In this regard, different from the rest, we have included an innovative hysteresis set of rules that allows the prediction system to minimize the number of false positives and generate a more reliable predicted value.

3 System Overview

In this section, we describe the 5G system used in this work along with a functional description of each of the building blocks present, discussing their importance in the forecasting method. In addition, we delve into the reason behind using ML techniques as a means to tackle traffic prediction and how it can be used as an enabler of network reconfiguration.

3.1 System Model

The system model envisioned in this work is depicted in Figure 1, which contains all the building blocks of the network presented. In particular, this system represents a virtualized 5G network aligned with the O-RAN vision, composed of the service management and orchestration framework, the network function layer and the key technical enablers. On the one hand, the infrastructure layer refers essentially to the cloud infrastructure, i.e., the Network Function Virtual Infrastructure (NFVI). On the other hand, the network function layer contains the basic 5G network stack, comprising the radio access network, the core network and the application servers. Finally, the key technical enablers contain the building blocks added in this work (together with the non-RT RIC) to the service management and orchestration framework, namely the System Orchestrator, the Slice Manager (SM), the telemetry and data collector, the message broker, and the ML Framework. It is worth highlighting that the elements introduced in the system model are fully compatible with the O-RAN architecture. The reason for this is that the AI/ML Framework and the telemetry collector represent a west interface towards the orchestrator, which interacts with the NFVI, if management operation are required. Therefore, no changes are introduced to the O-RAN elements and interfaces.

In this context, a network slice is provisioned through the Orchestrator and a Slice Manager, which deals with the compute resources and access network resources of the network slice, respectively. After the network slice is provisioned, telemetry data is periodically sent from a Telemetry Data Collector to a Message

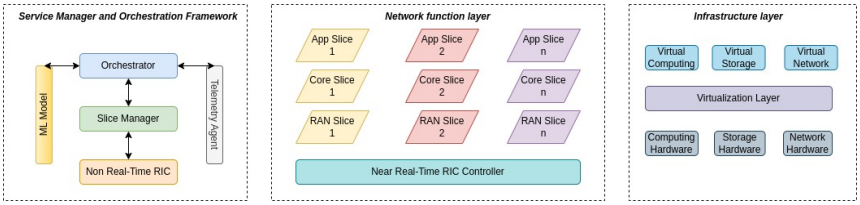

Fig. 1. Main building blocks of the proposed system model.

Broker. This data reflects the performance of the network slice, that is, the virtual elements (in this case Kubernetes Pods), and the physical servers where the pods are running.

A more detailed figure of the architecture is illustrated in Figure 2, where the joint vision of the network and compute resource orchestration, and actuation over network infrastructure/components, together with the AI engine is provided. The next subsections describe in detail the relationship between the various building blocks.

The selected telemetry data is sent to the ML Framework, where it is processed, analyzed and shared with the specific ML model handling this management operation to detect a possible alarm concerning the under/over-utilization of computational resources. This alarm might be caused by a sudden increase in traffic between the User Equipment (UE) and the applications running in the core. The main objective of the ML model in this work is to anticipate the increase or decrease in the incoming CPU capacity, so the network can adjust the virtualized resources accordingly. In case an alarm is predicted, the ML model informs the Orchestrator about it, which triggers the reconfiguration of the computational resources, guaranteeing the conditions agreed upon such slice and providing a tailor-made distribution of the network resources. For more information regarding the ML model, the reader is referred to Section 4.

3.2 Slicing Implementation

The slicing model envisioned in this work involves the allocation of both computing and radio resources in an End-to-End (E2E) manner, together with the provisioning of the required Virtual Network Functions (VNFs) for the network functioning. The SM is the module responsible for this task, and a key element in the 5G network architecture presented in this paper. It allows three key points: (i) seamless management of the network resources required by each network slice; (ii) deployment of services and applications for different verticals within the slices; and (iii) creation and management of network slice instances. In the 5G architecture studied in this paper, the SM acts as a proxy between the orchestrator (in charge of the computing -or infrastructure- resources, referred also as slice computing chunk) and the non-RT RIC (in charge of the radio resources, also named slice radio chunk) to allocate the resources needed by a slice.

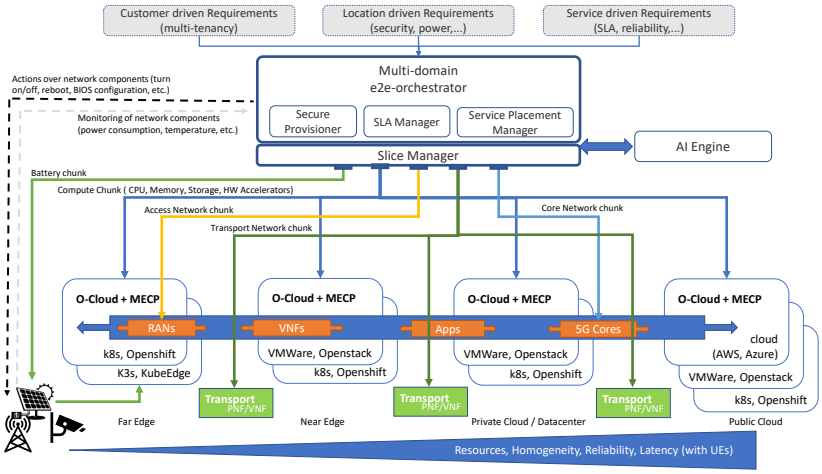

Fig. 2. Detailed network architecture overview.

Furthermore, the Slice Manager is responsible for the configuration of the core network and the initiation of the small cells, which is an essential step to provide full connectivity to any UE.

3.3 Orchestration Layer

The orchestrator is responsible for managing the Network Service (NS) life-cycle, along with the VNF life-cycle, supported by the VNF Manager (VNFM), and orchestrating Network Function Virtualization Infrastructure (NFVI) resources supported by the Virtualization Infrastructure Manager (VIM) to ensure an optimized allocation of the necessary resources and connectivity. Therefore, the orchestrator's functions can be classified into two main categories: E2E resource orchestration, and NS orchestration. The orchestrator is also responsible for guaranteeing adequate NS performance and fault management, as well as VNF package management.

To perform the actions described above, the orchestrator is composed of three main modules: (i) the secure provisioner, which provisions the node (e.g., installing the operating system, Hardware acceleration drivers, etc.); (ii) the SLA manager, which is responsible for guaranteeing the SLAs of the different tenants; and (iii) the service placement manager, which decides the optimal locations for the services to be executed. In addition, the orchestrator receives different kinds of requirements and information, i.e., from customer-driven requirements to location-aware information (e.g., power constraints) and service-level KPIs. Based on this information, the orchestrator can communicate with external entities (e.g., SM or an ML engine) to make the reconfiguration actions during the service lifecycle.

The reconfiguration actions regarding the network slice can be achieved through the ML control loop. To this end, the orchestrator monitors and collects all the monitoring data, KPIs, and logs exposed by the different components of the slice. This data is published in a telemetry time-series database (such as Prometheus) that can be used as input by an ML model. An ML model fed with this data is trained to raise alarms, which are published in a message broker. The orchestrator allows the user the definition of rules that, based on these alarms, enforce proactive reconfiguration of the network slice before an event happens.

3.4 Non-RT RIC

The Non-real-time RAN Intelligent Controller (Non-RT RIC) is a functional element defined by the O-RAN reference architecture [19], in charge of enabling the configuration of the disaggregated radio functions, i.e., Centralized Unit (CU) and Distributed Unit (DU), through the O-RAN O1 interface, as well as managing radio related policies deployed in the Near-real-time RIC (Near-RT RIC) through the A1 interface. In this work, we focus on automating radio configurations using the O1 interface. Moreover, dedicated network slices are provisioned leveraging the Multi-Operator Core Network (MOCN) functionality available in the base stations. In essence, deploying a new slice consists of deploying a dedicated core network instance, using the orchestrator and configuring the base stations that are part of that slice to add the core network's Public Land Mobile Network ID (PLMNID) to their list of radiated PLMNIDs and to include the IP address of the deployed core networks in its list of MOCN adjacency.

3.5 Telemetry Data Collector

The Telemetry Data Collector oversees the monitoring of both physical and virtual resources of the network slice. The telemetry data collected is made available for the ML algorithm to improve specific KPIs of the network. As mentioned in Section 3.3, the Orchestrator publishes on this element the telemetry data.

The Telemetry Data Collector can be viewed as a real-time data collection from various components of the architectural layers. Therefore, two relevant components are the Network Data Analytics Function (NWDAF) and the Centralized Management Data Analytics Function (C-MDAF), where the first one is a well-specified component of the 5G Core network according to 3GPP specifications. The NWDAF collects data from and provides network data analytic services to 5G core Network Function (NF). The C-MDAF is provisioned with all the centralized telemetry capabilities, located at the Management, Orchestration and Automation layer. In this context, a particular NF can subscribe to the C-MDAF as a consumer to collect or provide management data for forecasting or resource information purposes.

4 ML Pipeline in the Networking System

In this section, we describe the ML pipeline proposed in the system. In particular, we first discuss the rationale of the ML model built for the proactive adaptation of virtualized resources in mobile networks using forecasting models, covering the design proposed and the training process. Then, we describe the ML framework embedded in the network to facilitate not only the development but also the native deployment of AI/ML models.

4.1 Conceptual Design

The objective pursued in this work focuses on the design of a closed control loop able to reconfigure the virtualized computation capacity of a networking system. This work envisions an O-RAN-like mobile network as the one described in Section 3, in which several network slices with distinct requirements are deployed E2E, from the core to the radio units. The layout of the various slices is depicted in Figure 3, and as can be observed there, each slice is fully isolated from the RAN and core network perspective. All the slices leverage the NFVI resources, as specified in their SLA, and consume the services offered by their respective deployed applications.

Taking as a reference the above definition, we aim to analyze the KPIs of the network infrastructure and the applications deployed in each of the slices that have a direct relationship to the use of computational resources, and more in particular, to the CPU consumption of the slices. Based on this analysis, the main goal is to design a forecasting ML model able to anticipate the events causing under- and over-provisioning of the resources allocated to slices. Moreover, the ML model must proactively provide customized resources for every slice's needs, while helping guarantee optimized use of the network resources.

4.2 ML Pipeline Hosting

The increasing demand for AI/ML control loops requires dedicated architectural building blocks to facilitate all the different ML model development operations, including the data ingestion, training, evaluation and execution of the ML algorithms at every network level. To that end, we have developed an AI/ML framework based on open libraries and standards, including a set of open interfaces for integration with all the different network components of the architecture. The AI/ML framework presented in this paper is based on TensorFlow [20], the most widely adopted open-source set of libraries and toolkits for numerical computation and large-scale machine learning. TensorFlow ingests the data in the so-called tensors, which are multidimensional arrays of high dimensions, allowing TensorFlow to easily handle large amounts of data.

The architecture of the proposed AI/ML Framework is detailed in Figure 4. The first building block is the AI/ML Pipeline Orchestration Platform (POP), which is based on Apache Airflow [21] and allows the AI/ML developer to programmatically and sequentially run, schedule and monitor AI/ML pipelines in

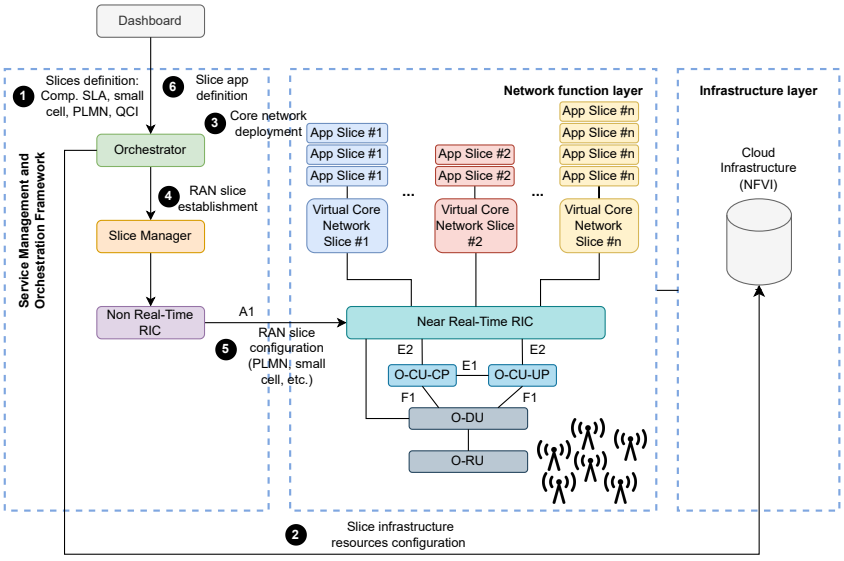

Fig. 3. Layout of the network slices for the ML training process.

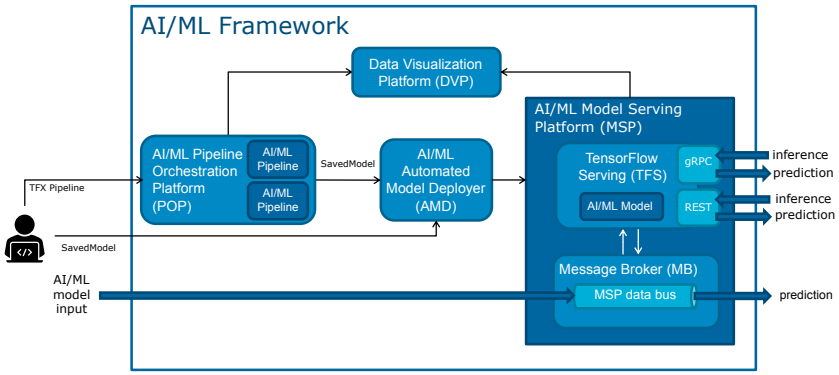

Fig. 4. AI/ML Framework architecture

the form of Direct Acyclic Graphs (DAGs). A DAG is a collection of tasks you want to run, properly organized and connected, reflecting relationships and dependencies.

The second building block is the AI/ML Automated Model Deployer (AMD), which interfaces the POP with the serving infrastructures. The AMD is in charge of creating a serving instance of new models on the AI/ML Model Serving Platform (MSP) or update already served models to a newer version. The MSP is the

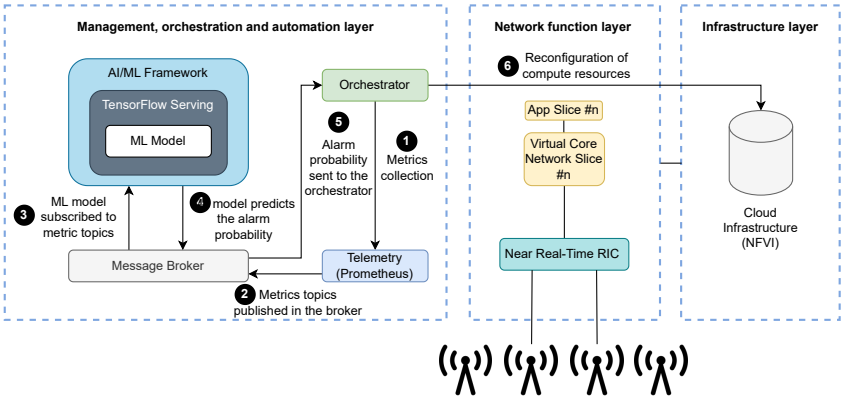

Fig. 5. Workflow of the ML model introduced in the network system.

building block in charge of serving the ML models. It includes a Message Broker (MB) based on RabbitMQ [22] and an instance that enables the simultaneous serving of multiple models, managing the versioning and model labeling, and enabling the separation of testing and production environments.

In the ML network reconfiguration scheme described in this paper, we have used the MSP data bus of the MB, as depicted in Fig. 5. Following this architecture, the *ML model's input* of the MB interfaces with the Prometheus instance using the Prometheus REST API [23]. This enables the ML model to periodically retrieve the (real-time) telemetry data collected. Conversely, the output of the ML model, which represents a probability of alarm, is periodically published by the MB on the MSP data bus. Consequently, the orchestrator receives the output from the ML model to initiate network infrastructure reconfiguration when necessary directly interacting with the NFVI.

The last building block of the AI/ML Framework is the Data Visualization Platform (DVP). It is formed by Prometheus [24] and Grafana [25] and allows the user to monitor real-time metrics from the POP and the MSP, including basic metrics like resource utilization, inference requests per model, incoming and outcoming messages in the MSP data bus, etc.

All the aforementioned building blocks have been developed as Docker containers that can be easily deployed in a Kubernetes environment. This fact makes this solution highly portable and scalable, enabling the effortless orchestration of each element individually scaling the resources on demand, and replicating the containers for load balancing and reliability purposes.

We acknowledge that in a fully deployed, production-level O-RAN architecture, the model would exclusively interact with the Orchestrator. The Orchestrator would then communicate with other nodes in accordance with established standard protocols, such as O1, O2, and A1 interfaces, as specified by the O-RAN Alliance and 3GPP. Consequently, there are no other interactions between

the proposed ML model and any other entities in the O-RAN architecture, as the output of the ML model does not affect the management and control of the radio nodes. The O-RAN Working Group 2 has standardized the ML/AI workflow to facilitate the interaction between ML models and the specific O-RAN architecture, as described in [26]. Following this scheme, data is acquired from the O1, A1, and E2 interfaces and stored in extensive datasets, typically centralized repositories referred to as data lakes. These repositories serve as sources for data storage, enabling extraction when required. Our ML model requires specific configuration of input data, and to fulfill this requirement, the O-RAN specifications include an initial step of data pre-processing or preparation. During this step, the collected data, both for training and online inference purposes, undergoes shaping and formatting to align with the input size of the AI/ML model under consideration. After completing the inference stage, the ML model sends the necessary actions to the corresponding entity. Configuration management tasks are executed via the O1 interface, policy management is facilitated through the A1 interface, and control actions or policies are transmitted via the E2 interface.

It is important to note that if the capabilities of this ML model are extended in the future to also account for the management of the radio resources allocated to a given slice, the orchestrator would communicate this decision to the Non-RT RIC using the E2 interface, which would then take the corresponding actions on the radio configuration.

4.3 Data Preprocessing

An extensive process has been performed before the model training, with two main objectives: (i) online data acquisition, and (ii) offline data preprocessing. The testbed used for this data acquisition step is described in Section 5.1.

The data acquisition process has been carried out online by deploying an Iperf server as the application requested on a particular instantiated slice, and an Iperf client on the UE connected to such a slice. This choice in the application seeks to enable in the slice a variety of bitrates generated by various traffic types, including TCP and UDP. This application has been deployed by the network orchestrator on a container, together with a Python script that is in charge of tuning the parameters used by Iperf for the data collection. The data collection has been organized into different stages on which the aforementioned traffic parameters are modified, gathering a dataset with a total duration of around 8 hours. The traffic injected, as well as the virtual infrastructure resource KPIs, have been retrieved every 5 seconds by using a Prometheus server deployed along with the VIM.

At the beginning of the data collection process, the server has been initialized with the lower bound of the bitrate range envisioned (10 Mbps), while the script is additionally responsible for selecting with a 50% probability whether the traffic generated is TCP or UDP. The use of two types of traffic aims to show if, in addition to the bitrate injected, the transport protocol employed also impacts the network resources. This bitrate is maintained for 180 seconds, and then

randomly increased or decreased in the range of [10 - 100 Mbps]. This process of modifying the bitrate value is performed five times after which the script forces the Iperf application to select again the traffic type (i.e., TCP, UDP) to be used and restarts the operation.

Initially a total of 18 network parameters have been obtained as a result of the data collection. Six of these parameters are linked with the status of the Kubernetes node in which the slice is allocated, while the remaining twelve are bound to the infrastructure dedicated to the application of the slices. This Prometheus output has been parsed to obtain a time-series dataset that can be used for training and validation purposes.

Following the data collection, an important preprocessing step is to discern which of these collected parameters have a direct impact on future values of the CPU usage. This step has been carried out by using the Granger causality test [27], which is a statistical hypothesis test designed to determine if a time series can be used to forecast another. This test has been performed to find the subset of available parameters that can be useful to predict future values of the *container CPU usage seconds* variable. Results have indicated a subset of 7 of these parameters, namely *node memory percentage, node long-term load, node midterm load, container file system usage bytes, container memory usage bytes, container network transmitted bytes*, and *container CPU usage seconds* itself.

We have further reduced the number of parameters to use for the CPU usage forecasting, limiting them to two, which represent the input of the ML model: *container network transmitted bytes* and past values of the *container CPU usage seconds* itself. This selection has been made after some manual experimental and visual analysis of the 7 preselected parameters. Both selected features have been found to affect in a major way the status of the CPU in future time steps.

4.4 ML Model Description

In this paper, we use Neural Networks (NN) for predicting the computational resources of a slice [28]. The rationale behind the adoption of NNs is twofold: (i) the use of NNs substantially reduces the manual actions related to both control and management of 5G networks [29]; and (ii) compared with other non-linear forecasters, NNs provide higher accuracy levels. Specifically, the ML model presented in this work is based on a Feedforward Neural Network (FNN), which is a model where information flows in only one direction; this is, from the start node going through the hidden layer nodes until the final nodes (without any type of feedback or closed loops).

The FNN employed in this work is composed of one input layer with 30 neurons, followed by a hidden layer of 64 neurons and finally an output layer with a sigmoid activation function. The input of the FNN is obtained from two Prometheus metrics (as described in Section 4.3): *CPU usage seconds* and *container network transmitted bytes*. 15-time samples of these two metrics are gathered, and each sample is linked to an individual neuron. Therefore, 30 neurons in total are used as inputs of the model. Several time sample numbers have

been tested and used, but the best results are obtained while using 15 as the time sample number.

The value of the output layer of the ML model represents the probability (i.e., a value between 0 and 1) of having an alarm in the next 30 samples in a network slice. An alarm is defined as a sudden and drastic increase in CPU consumption. This can occur due to an increase or decrease in traffic from the UE to the application running in the containers and/or due to bad planning regarding the computational resources allocated to the slice. If the probability exceeds a predefined threshold value, the ML model's output will trigger an alarm, prompting the orchestrator to adjust the computational resources within the NFVI of the slice. This adjustment aims to handle the incoming network traffic while ensuring the quality of service for established slices and improving network throughput. To further improve the accuracy of the FNN, the dropout is set to 10%. Therefore, in every batch size run (training the model using the defined batch size once), one out of ten randomly selected neurons is turned off. This random selection is performed to avoid the overfitting of the results, as when the NN works with different neurons off, it becomes more robust.

5 Experimental Evaluation

In this section, we describe the testbed used to evaluate the method proposed in this paper. Moreover, we discuss in detail the experimental results and their impact on the overall network throughput. To show how the ML closed control loop reconfiguration works and how the computational resources of the containers can be fine-tuned to each slice, we have defined a set of experiment settings, whose methodology is described in the section.

5.1 Real-world Testbed Description

The testbed introduced in this section is used for both data acquisition during the training phase of the ML model in Section 4, as well as for its validation and performance evaluation.

The physical deployment is located at the Castelloli Parcmotor test track, a circuit near the city of Barcelona (Spain) usually employed by carmakers for vehicle stress-testing and validation sessions. Two sites in the circuit were selected: the control room and one of the cell sites, connected through a microwave link used as backhaul of the network. The control room accommodates servers specifically designed for virtualized mobile networks at the Edge, namely Lenovo ThinkSystem SE350 Edge Servers [30]. In our experiments, we have used two of these servers. They were initially provisioned with CentOS as the base OS, and Kubernetes as the VIM. For the orchestration, we have installed the state-of-the-art NearbyOne platform [31], which can provision and monitor the bare-metal servers, CIM nodes and the apps running in containers. The monitoring data is stored in a Prometheus database and is available for the ML-Engine for predictive actions.

On the other hand, the 5G access layer is based on a virtualized Open RAN, including an Accelleran dRAX (version 2.1), and Small Cells model E1000. This radio has been demonstrated to be appropriate for various types of wireless network deployments: from sparsely populated areas to capacity-constrained and ultra-dense scenarios. This small cell can be easily configured to work in rural environments using high-gain directional antennas or in urban/suburban environments using omni-directional antennas. This second configuration is particularly useful when addressing capacity-constrained and ultra-dense deployments. The radio also supports most of the Time-Division Duplexing (TDD) and Frequency-Division Duplexing (FDD) bands, with a particular emphasis on the 3.5GHz worldwide mobile capacity in B42, B43, and the B48 CBRS/OnGo variant. For the specific deployment, the selected frequency band was 3400-3420 MHz with a 20 MHz carrier (B42 band) [32]. We have used the servers as the physical devices in which to instantiate the virtualized entities previously described (i.e., non-RT RIC, SM, Orchestrator and the Telemetry Data Collector), helping us also to deploy a real-world multivendor, disaggregated Open RAN with a 5G core based on the Druid Raemis 4G/5G solution, which is 3GPP-compliant, as described in [33].

There are two key advantages of deploying the scenario in a real-world location, following a 3GPP architecture standard: first, even if the data of the model is obtained in a test scenario, the results obtained are closer to a real commercial deployment of a 5G network, as opposed to just testing the model in an enclosed simulation environment; second, by testing the model in an architecture compliant with international specifications, we can guarantee the compatibility of the model with said standards and the possibility to escalate it on commercially launched architectures.

5.2 ML Model Evaluation

In the following lines, we describe the experiments and the rationale behind the design and tuning of the proposed ML model. It is worth mentioning that the values of the CPU usage (measured every 5 seconds) normally have high fluctuations over time. As a result, the accuracy of the prediction of CPU usage is not a trivial task. Another challenge is the quickness of the prediction since the ML model must predict the high increase in CPU usage as soon as possible to trigger the required reconfiguration actions, hence minimizing the negative impact on the network throughput. To cope with both accuracy and quickness, the training of the ML model has been tackled by creating two new variables:

- An *alarm* variable with values {0,1}. At each time step, this variable has value 1 if the value of the feature *CPU usage seconds* surpasses a pre-defined threshold, where the *CPU usage seconds* is defined as the CPU time consumed per CPU (for each Kubernetes POD) in seconds.
- An *alarm-next-30* variable with values {0,1}, where the value 1 means that there is at least one alarm within the next 30 time steps (i.e. at least one of the thirty following samples has the *alarm* variable set to 1).

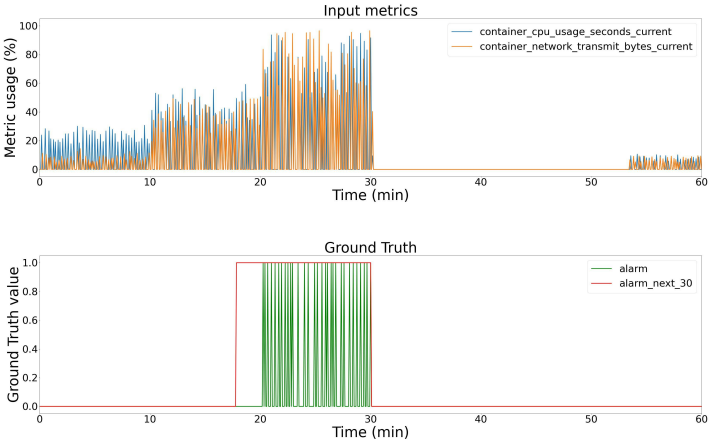

Fig. 6. Input variables and ground truth label used during training. The *alarm* variable is also represented for comparison purposes with the ground truth label.

The variable *alarm-next-30* has been introduced because it is more stable than the *alarm* variable. In this way, the initial problem of forecasting a CPU usage value has been adapted to a simpler binary classification task. Within this new task, the model takes as input the past values of both the *container network transmitted bytes* and *CPU usage seconds* KPIs and outputs a value indicating the probability of an alarm happening in the next 30 time steps. Notice that the purpose of the *alarm* variable is to create the *alarm-next-30* variable, but only the latter has been used during the training of the model. Figure 6 shows an example of both input variables during six minutes (360 seconds), the ground truth label and the *alarm* variable, where the x-axis represents the sample time (time steps). Notice that the *alarm-next-30* variable is much more stable in time than the single *alarm* variable. The container *network transmitted bytes* and the *container CPU usage* variables have been normalized to fit in the figure.

The training process of the FNN involved an iterative algorithm that adjusts the network parameters (weights and biases) to minimize the difference between the predicted output and the actual output of the network on the training data. This is achieved using a technique called backpropagation, which calculates the gradient of the loss function with respect to the network parameters and updates them accordingly using an optimization algorithm such as stochastic gradient descent (SGD).

Figure 7 shows the training process of the FNN. As we can see, the validation error is lower than the training error, usually an indication that the model is generalizing the information appropriately. Additionally, both errors tend to zero

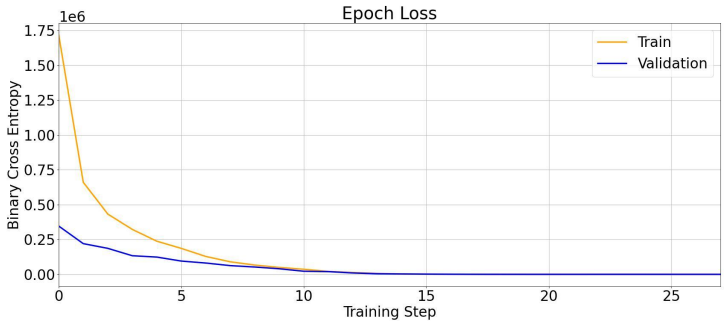

Fig. 7. Input variables and ground truth label used during training. The *alarm* variable is also represented for comparison purposes with the ground truth label.

in the long term, suggesting that the FNN model has learned to fit the data in a good way. We have used binary cross entropy as a loss function because our goal is to predict a binary output (0 or 1) based on the set of input features. The use of binary cross entropy as the loss function is motivated by the fact that it is well-suited for measuring the dissimilarity between the predicted and actual outputs, as it penalizes large deviations from the true values more heavily.

Once the ML model is trained, we have observed that it provides outputs that can be considered unstable over time, meaning that the output of the model can vary from nearly 1 to nearly 0 and vice versa between consecutive time steps. Therefore, a set of hysteresis rules have been defined to address the negative sudden variations in the ML output.

Every output of the model is stored in a buffer of historical predictions. This buffer works as a FIFO queue with a fixed size n, and its values are considered to make the final decision of future time steps. Similarly, the average of the *CPU usage seconds* variable in the last n *samples* is computed at each time step. To smooth out the noise in data, the average is calculated by replacing each data point with the average of its neighbor values in a moving kernel.

With the historical predictions buffer, the CPU average and, being *prediction* the output of the model at the current time step, the hysteresis rules are applied as follows.

- If $prediction \geq 0.8$ (i.e., high probability of alarm in the next 30 time steps), then the average CPU usage is checked. The final hysteresis prediction is set to 1 if this average is above a certain threshold, otherwise, it is set to 0. This rule helps to filter out false positive outcomes of the model when the CPU usage has no sign of stress. We must remark that the 0.8 threshold is based on extensive simulation results, where several experiments with different types of traffic were conducted.
- If $prediction < 0.8$ (i.e., low probability of alarm in the next 30 time steps), then the buffer of the historical predictions is checked. If there is at least

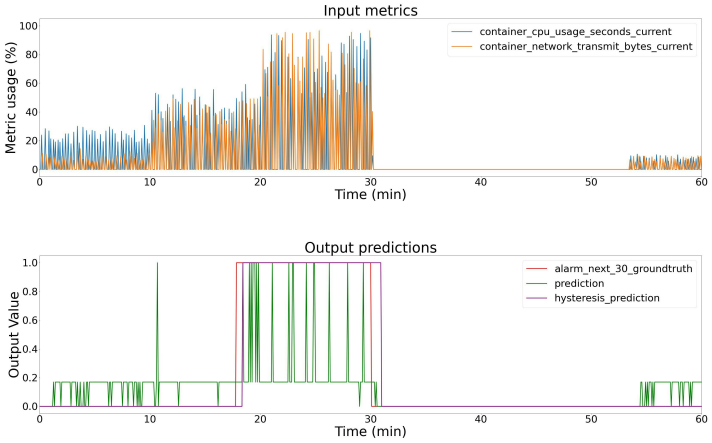

Fig. 8. Comparison between the ML raw output and the final ML prediction after applying the hysteresis rules.

one alarm prediction in the buffer, the final hysteresis decision is set to 1, otherwise, it is set to 0. In other words, it forces the model to output several no-alarm decisions consecutively before considering that an alarm is no longer present. This rule is useful for filtering out false negative outcomes when there is still a situation of alarm.

Figure 8 shows an example of the behavior of the ML model and the final decision once the hysteresis rules are applied. Notice that the raw output of the model is shown in color orange and the final prediction after applying the hysteresis rules is depicted in green. The alarm ground truth, which refers to the *alarm-next-30* is represented in red. Finally, the container CPU usage (blue color) has been normalized to fit in the figure.

Additionally, accuracy results on test data have been computed using the definition of *balanced accuracy score* given, as described in [34], by the following equation:

$$\frac{1}{2}\left(\frac{TP}{TP+FN}+\frac{TN}{TN+TP}\right) \tag{1}$$

where TP, TN, FP, FN correspond to the True Positive, True Negative, False Positive and False Negative predictions after applying the hysteresis results. Balanced accuracy is a much more accurate metric than normal accuracy when working with unbalanced datasets. This is exactly the case in this work, since there is a clear majority class (i.e., negative outcome meaning "no-alarm") and a minority class (i.e., positive outcome meaning "alarm"). In binary classification tasks such as the one in this paper, the balanced accuracy metric is equivalent

to the arithmetic mean of the true positive rate (or recall) and the true negative rate. Table 1 depicts all the aforementioned metrics. In this regard, we have drawn the following conclusions about the obtained results:

- The true negative rate indicates an overall great performance when it comes to the avoidance of false positive outcomes. Although there are still some of them, this issue can be further addressed, for example, by requiring a minimum number of consecutive positive outcomes before triggering an alarm.
- The true negative rate shows that there is still room for improvement to reduce the number of false negative outcomes. However, we have observed that all of the "alarm" CPU usage peaks have been correctly detected, although not with 30 samples of anticipation as the ground truth requires.

Table 1. Accuracy metric results of the ML model on test data.

Raw Accuracy	Balanced Accuracy	True Positive Rate (Recall)	True Negative Rate
0.980	0.915	0.839	0.991

5.3 Impact of the ML Model on the 5G Network Performance

To properly evaluate the benefits of the proposed ML model regarding the overall network performance (specifically on the network throughput), we conducted the following experiments. We deployed two different slices in our testbed. Each of these slices has its pool of resources assigned (compute chunk, access network chunk, core network chunk).

The full life cycle and configuration of the different components is managed by the Orchestrator. From the bottom up, the server's HW configuration, its operating system and the deployment of the containerized RAN, core, applications, and slices are managed and monitored by the Orchestrator. The slices configured for the experiment are assigned limited CPU resources. We remind the reader that, in this work, capacity forecasting is specifically focused on the CPU resources assigned to the compute chunk. In this regard, and to show the bandwidth degradation and the impact of dynamic slice provisioning, only the second slice is monitored by the ML-powered closed-control loop. By contrast, to focus on the ML model's decisions regarding the infrastructure resources, and be able to extract proper conclusions, we have dimensioned the radio access and core network chunk's resources large enough, so that they do not constitute a bottleneck for the bandwidth values considered in the experiments.

Notice that for the first slice, which has fixed computational resources, and does not have the ML closed control loop, we observe severe bandwidth service degradation (65 Mbps) when the CPU resources assigned to the compute

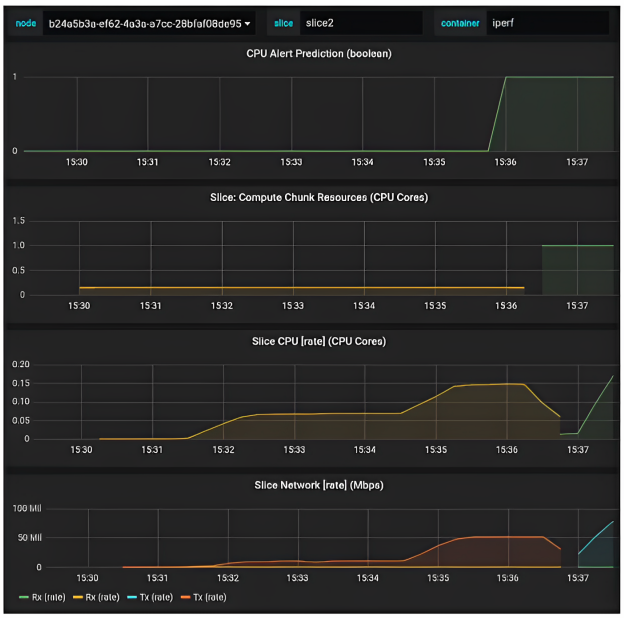

Fig. 9. Grafana snapshot with a triggering event example.

chunk become the bottleneck. Hence, the UE is not able to achieve its maximum throughput. However, for the second slice (which has the ML features), the ML closed control loop triggers the resizing of the compute chunk. As a result, the Orchestrator deploys a new pod with increased CPU resources. Only after the new pod is running, the first instance is removed and the Iperf download can switch to the new Iperf server. This will enable the UE to achieve its maximum network throughput.

Figure 9 shows a triggering event from the point of view of the orchestrator and its Grafana dashboard. On the top row, under *CPU Alert Prediction*, it can be seen how at time 15:36 the ML-Engine algorithm identifies an alarm pattern that triggers the rule in the Orchestrator. Hence, the consequent pod for this slice is resized from 0.2 to 1.0 CPU cores. At the bottom row in the same figure, i.e., *Slice Network RTx (Mbps)*, it can be observed that after the iperf client detects the change, it reconnects to the new pod. During that transient period, the bandwidth falls initially, but it is able to recover and increase the previously obtained bandwidth. This transient period, where performance seems to degrade, can be explained by the fact that we are using a simple iperf client script with no kind of support to handle these updates proactively. Until the initial iperf connection timeouts, it does not try to reconnect. In this sense, it is possible to state that the proactive reconfiguration of the resources required

in the compute resources of the slice results in an increase of the network slice performance, reflected by an almost 4x enhancement (as depicted on the bottom of Fig. 9).

6 Conclusions

In this paper, we introduced an ML model capable of successfully forecasting the over-utilization of computational resources in virtualized 5G networks. This ML model has the potential to be a key asset that can be leveraged for network reconfiguration actions in future virtualized wireless networks, providing the ability to properly fulfill the QoS of the slices while providing a way to optimize the network resources. Our test experiment was conducted in a real testbed located in Castelloli, Spain, which is based on a virtualized 5G wireless network and a Kubernetes infrastructure using baremetal servers.

The results obtained indicate that a Fully Connected Neural Network with a single hidden layer and some simple post-processing hysteresis rules can accurately detect the saturation of computing resources, achieving up to 91% balanced accuracy and close to 84% recall (true positive rate) for both TCP and UDP traffic. Therefore, this paper demonstrated: (i) the impact that network traffic has on the computational resources allocated to a slice on the obtained throughput; and (ii) the successful use of an ML-based prediction model that supports the automatic reconfiguration of the computational resources. Notably, the performance results obtained are independent of the number of slices, as the ML model's prediction is executed separately for each slice.

It is important to note that, due to the flexible capacity of the ML model used in the prediction of the CPU alarm, the analyzed feature can be easily replaced. This means that with some small changes, the model would be able to predict another feature of the network without any core changes in its design, enforcing its adaptability and flexibility.

As a future line of work, we plan to extend the proposed ML model to go a step further regarding slice reconfiguration. To this end, we envision an ML model capable of detecting not only the saturation of computational resources, but also of providing as output: (i) the amount of computational resources that must be reassigned; and (ii) the most suitable Kubernetes container from which resources can be allocated according to application requirements and constraints, enhancing the ability of the network to provide QoS to specific slices or services.

7 Acknowledgement

This work has been partially supported by the Affordable5G project, funded by the European Commission under Grant Agreement H2020-ICT-2020-1, number 957317 through the Horizon 2020 and 5G-PPP programs (www.affordable5g.eu). The authors would also like to acknowledge CERCA Programme / Generalitat de Catalunya for sponsoring part of this work. This work has been also supported by the EU "NextGenerationEU/PRTR", MCIN and AEI (Spain) under project

IJC2020-043058-I. The authors would also like to acknowledge the fundings of NANCY, number 101096456 through the Horizon Europe project.

8 Declarations

Ethical Approval. Due to the characteristics of the experiments performed in this paper, this declaration is not applicable.

Competing interests. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. M. Giordani, M. Polese, A. Laya, E. Bertin, M. Zorzi, 6g drivers for b2b market, in: Shaping Future 6G Networks: Needs, Impacts, and Technologies, 2022.
2. O-RAN Architecture, `https://www.o-ran.org`, [Online; accessed May-2023]. (January-2022).
3. T. Panagiotis, L. Sarakis, A. Giannopoulos, S. Spantideas, N. Capsalis, P. Gkonis, K. Panagiotis, G. Rigazzi, A. Antonopoulos, M. Cambeiro, S. González, L. Conceição, A Cost-Efficient 5G Non-Public Network Architectural Approach: Key Concepts and Enablers, Building Blocks and Potential Use Cases, Sensors 21 (16) (2021) 55–78.
4. D. Bega, M. Gramaglia, A. Garcia-Saavedra, M. Fiore, A. Banchs, X. Costa-Perez, Network slicing meets artificial intelligence: an AI-based framework for slice management, IEEE Communications Magazine 58 (6) (2020) 32–38.
5. F. Ramos, E. Viegas, A. Santin, P. Horchulhack, R. R. dos Santos, A. Espindola, A machine learning model for detection of docker-based app overbooking on kubernetes, in: Proc. of IEEE ICC, 2021.
6. M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, G. Wunder, 5G: A tutorial overview of standards, trials, challenges, deployment, and practice, IEEE Journal on Selected Areas in Communications 35 (6) (2017) 1201–1221.
7. Y. Cho, S. Jang, S. Pack, On Performance VNF Load Prediction Models in Service Function Chaining, in: Proc. of IEEE ICTC, 2020, pp. 344–346.
8. R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, R. Boutaba, Topology-aware prediction of virtual network function resource requirements, IEEE Transactions on Network and Service Management 14 (1) (2017) 106–120.
9. H.-G. Kim, D.-Y. Lee, S.-Y. Jeong, H. Choi, J.-H. Yoo, J. W.-K. Hong, Machine learning-based method for prediction of virtual network function resource demands, in: Proc. of IEEE NetSoft, 2019, pp. 405–413.
10. P. Tam, S. Math, S. Kim, Priority-Aware Resource Management for Adaptive Service Function Chaining in Real-Time Intelligent IoT Services, Electronics 11 (19) (2022) 2976.
11. Q. Xia, W. Ren, Z. Xu, P. Zhou, W. Xu, G. Wu, Learn to optimize: Adaptive VNF provisioning in mobile edge clouds, in: Proc. of IEEE SECON, 2020, pp. 1–9.

12. L. Toka, G. Dobreff, B. Fodor, B. Sonkoly, Machine learning-based scaling management for kubernetes edge clusters, IEEE Transactions on Network and Service Management 18 (1) (2021) 958–972.
13. S. Sengupta, J. Garcia, X. Masip-Bruin, Collaborative learning-based Schema for Predicting Resource Usage and Performance in F2C Paradigm, in: Proc. of IEEE ANTS, 2020.
14. S. K. Singh, R. Singh, B. Kumbhani, The evolution of radio access network towards open-ran: Challenges and opportunities, in: Proc. of IEEE WCNCW, 2020, pp. 1–6.
15. M. Polese, L. Bonati, S. D'Oro, S. Basagni, T. Melodia, Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges, IEEE Communications Surveys & Tutorials (2023) 1–1.
16. P. Li, J. Thomas, X. Wang, A. Khalil, A. Ahmad, R. Inacio, S. Kapoor, A. Parekh, A. Doufexi, A. Shojaeifard, R. J. Piechocki, RLOps: Development Life-Cycle of Reinforcement Learning Aided Open RAN, IEEE Access 10 (2022) 113808–113826.
17. M. Karbalaee Motalleb, V. Shah-Mansouri, S. Parsaeefard, O. L. Alcaraz López, Resource Allocation in an Open RAN System Using Network Slicing, IEEE Transactions on Network and Service Management 20 (1) (2023) 471–485.
18. B. M. Xavier, M. Dzaferagic, D. Collins, G. Comarela, M. Martinello, M. Ruffini, Machine learning-based early attack detection using open ran intelligent controller, arXiv preprint arXiv:2302.01864 (2023).
19. Non-RT RIC, `https://docs.o-ran-sc.org/projects/o-ran-sc-nonrtric/en/cherry/overview.html`, [Online; accessed May-2023]. (January - 2023).
20. TensorFlow: Large-scale machine learning on heterogeneous systems, `https://www.tensorflow.org/` (2015).
21. Apache AirFlow, `https://airflow.apache.org/`, [Online; accessed May-2023]. (January - 2023).
22. RabbitMQ, `https://www.rabbitmq.com/`, [Online; accessed May-2023]. (January - 2023).
23. Prometheus HTTP API, `https://www.prometheus.io/docs/prometheus/latest/querying/api/`, [Online; accessed May-2023]. (January - 2023).
24. Prometheus NMS, `https://prometheus.io`, [Online; accessed May-2023]. (2023).
25. Grafana, `https://grafana.com/`, [Online; accessed May-2023]. (January - 2023).
26. O.-R. Alliance, O-ran working group 2, ai/ml workflow description and requirements, O-RAN ALLIANCE (2021).
27. C. W. J. Granger, Investigating Causal Relations by Econometric Models and Cross-Spectral Methods, Econometrica 37 (3) (1969) 424–438.
28. S. C. Kleene, Representation of Events in Nerve Nets and Finite Automata, Princeton University Press, 2016, pp. 3–42.
29. D. Ferreira, A. Braga Reis, C. Senna, S. Sargento, A Forecasting Approach to Improve Control and Management for 5G Networks, IEEE Transactions on Network and Service Management 18 (2) (2021) 1817–1831.
30. Lenovo servers, `https://www.lenovo.com/us/en/p/data-center/servers/edge/thinksystem-se350/77xx6dsse35`, [Online; accessed May-2023]. (January - 2023).
31. NearbyOne Orchestrator, `https://www.nearbycomputing.com/nearbyone/`, [Online; accessed May-2023]. (February 2022).
32. drax solution, `https://accelleran.com/drax-2/`, [Online; accessed May-2023]. (January - 2023).
33. Druid 5g core solution, `https://www.druidsoftware.com/raemis-cellular-network-technology/`, [Online; accessed May-2023]. (January - 2023).
34. L. Mosley, A balanced approach to the multi-class imbalance problem, Ph.D. thesis, Iowa State University (2013).