Service-Aware Real-Time Slicing for Virtualized beyond 5G Networks

Theodoros Tsourdinis, Ilias Chatzistefanidis, Nikos Makris, Thanasis Korakis, Navid Nikaein, Serge Fdida

Graphical Abstract

Service-Aware Real-Time Slicing for Virtualized beyond 5G Networks

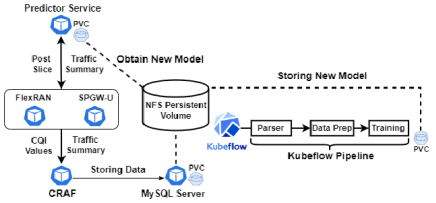
Theodoros Tsourdinis, Ilias Chatzistefanidis, Nikos Makris, Thanasis Korakis, Navid Nikaein, Serge Fdida

Problem Statement

- Increasing demand for efficient resource allocation in 5G networks.
- Challenges in adapting to dynamic application usage and diverse user requirements.
- Addressing the dynamic lifecycle of ML models in 5G network slicing.

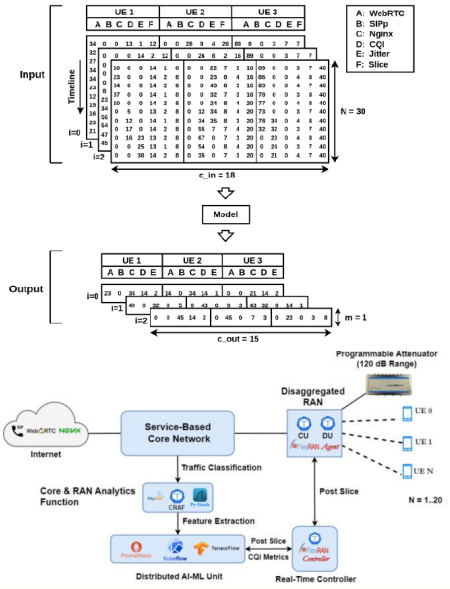
Contribution

- Innovative solution for dynamic application-aware network slicing.
- Leveraging 5G architecture, developing MLOps methodologies for distributed and online training to ensure continuous model refinement and adaptation.



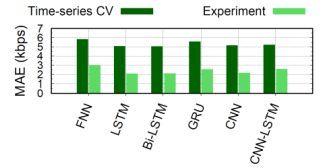
Approach

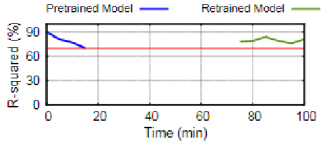
- Traffic classification from the network edge, analyzing and predicting demands from services running within the RAN
- Utilized machine learning for intelligent resource optimization.

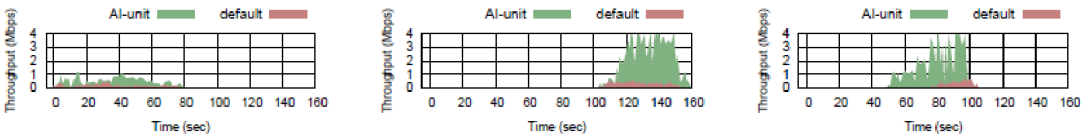


Outcomes

- Evaluation in a testbed setup, with realistic data traces replicated by using signal attenuators
- Using open-source platforms for experimentation (OpenAirInterface & FlexRAN)
- Experimental comparison of six different ML models for predicting traffic demands and selecting the best performing one.
- Results denote that for our experimental settings, applications running over the network can get up to 4 times higher network capacity when needed.
- Improved model accuracy and adaptability through MLOps-driven distributed/online training, fostering robust and scalable 5G network slicing solutions.







ELSEVIER

Computer Networks, Network Softwarization and Intelligence at the Edge: Challenges and Opportunities

Theodoros Tsourdinis, Ilias Chatzistefanidis, Nikos Makris, Thanasis Korakis, Navid Nikaein, Serge Fdida

Highlights

Service-Aware Real-Time Slicing for Virtualized beyond 5G Networks

- Developed a custom Network Data Analytics Function (NWDAF) function to parse statistics from both core and RAN.
- Introduced an MLOps architecture for online and distributed training via the cloud across cluster nodes.
- Collected dataset metrics including Jitter, Throughput for user experience, and CQI from realistic mobility scenarios using programmable attenuators.
- Applied a deep learning approach to enhance resource allocation (slicing).
- Tested the framework's performance using six distinct deep-learning models.
- Demonstrated that our proposed scheme, in contrast to a non-AI baseline unit, significantly excels in both efficiency and resource allocation, reducing over and under-provisioning.

Service-Aware Real-Time Slicing for Virtualized beyond 5G Networks

Theodoros Tsourdinis[a,b], Ilias Chatzistefanidis[c], Nikos Makris[a], Thanasis Korakis[a], Navid Nikaein[c], Serge Fdida[b]

[a] *University of Thessaly, Dept. of ECE, Volos, Greece*
[b] *Sorbonne Université, CNRS, LIP6, Paris, France*
[c] *EURECOM, Sophia-Antipolis, France*

Abstract

Edge Intelligence is expected to play a vital role in the evolution of 5G networks, empowering them with the capability to make real-time decisions regarding various allocations related to their management and service provisioning to end-users. This shift facilitates the transition from a network-aware approach, where applications are developed to manage network quality fluctuations, to a service-aware network that self-adjusts based on the hosted applications. In this paper, we design and implement a service-aware network managed from the network edge. We utilize and assess various Machine Learning models to classify cellular network traffic flows in the backhaul, aiming to predict their future impact on network load. Leveraging these predictions, the network can proactively and autonomously reallocate slices in the Radio Access Network via programmable APIs, ensuring the demands of the traffic-generating applications are met. The approach integrates innovative MLOps methodologies for distributed and online training, enabling continuous model refinement and adaptation to evolving network dynamics. Our framework was tested in a real-world environment with realistic traffic scenarios, and the results were evaluated in real-time, down to a granularity of 10ms. Our findings indicate that the network can swiftly adjust to traffic, providing users with slices tailored to their application needs. Notably, our experiments show that under the studied settings, the users experienced up to 4 times lower latency (jitter) and nearly 4 times higher throughput when interacting with various applications, compared to the standard non-AI/ML unit. Furthermore, our dynamic scheme significantly optimizes resource allocation, ensuring energy efficiency by avoiding over- and under-provisioning of resources.

Keywords: Beyond 5G, Service-Aware, RAN Slicing, OpenAirInterface, Kubernetes, Machine Learning, MLOps

1. Introduction

Edge Intelligence is widely considered the key element for empowering innovation and enabling the beyond 5G and future 6G networks to meet their full potential. It is expected that within 6G, edge intelligence will enable networks to achieve massive performance gains through unique functions and services that take advantage of the close proximity to the Radio Access Network (RAN), while re-program the network operation through the available APIs (e.g. O-RAN for the RAN). Artificial Intelligence is thus playing a major role in this context, allowing the transformation from network observations to key decisions that affect the overall system performance and reliability, even under high traffic loads.[1] Such decisions are fortified through the Multi-access Edge Computing (MEC) architecture, enabling low-latency applications to be hosted over the network with traffic breaking out from the edge to any Data Network (DN) [2].

The cornerstone for all these innovations is the wide softwarization that has taken place in 5G and beyond networks; services that up to the 4th generation were running as monolithic components, locked in vendor-specific hardware, are currently able to be hosted over generic hardware, running as software network functions. The components have been further disaggregated, by specifying standardized interfaces for their intercommunication, realizing a full Service Based Architecture (SBA), capable of instantiating in a cloud-native manner. This approach extends even for the cases of the RAN, for the higher level functions of the base stations, that can be realized through software functions placed on the edge/cloud, communicating with the Radio Units through high capacity fronthaul links (Cloud-RAN) [3]. The combination of all these features, empowered by Edge Intelligence, creates fertile ground for introducing novel services that manage the virtualized cellular network even in real-time/near-real-time.

Network slicing is a fundamental concept in 5G networks. It refers to the process of creating multiple virtual networks on top of a shared physical infrastructure. Each "slice" is tailored to meet the specific requirements of a particular service or application, ensuring optimal performance and resource utilization. Although such inno-

Email addresses: ttsourdinis@uth.gr (Theodoros Tsourdinis), ilias.chatzistefanidis@eurecom.fr (Ilias Chatzistefanidis), nimakris@uth.gr (Nikos Makris), korakis@uth.gr (Thanasis Korakis), navid.nikaein@eurecom.fr (Navid Nikaein), serge.fdida@sorbonne-universite.fr (Serge Fdida)

vations allow the efficient provisioning of network service under one/more slices with guarantees, usually it is up to the hosted applications to self-adapt to the fluctuations of the network service. For example, in the case of adaptive video streaming, protocols like DASH [4] might request the specific content that can be served over the network, based on the application perception of the network settings (e.g. capacity, jitter, delay, etc.). The disaggregation of network functions, as it has been standardized for 5G, enables the development of further key *xApps* that can take advantage of the APIs, allowing the network to self-adapt based on the applications that are hosted over the top, through the decisions for allocation in the network. Such decisions are usually based on the spectrum allocation (e.g. for Dynamic Spectrum Management [5]), or slicing allocation. In this work, we deal with the slicing part of the network, for automating the slice allocation of the network, based on the services that run on top, thus creating a fully-fledged *service-aware* network.

The development of such functionalities relies heavily on resource disaggregation as defined for 5G networks. This disaggregation has been standardized for different parts of the network (Control/Data Plane and RAN/Core Network) as follows: 1) RAN disaggregation for the base station stack, based on the eight different 3GPP defined functional splits [6], and 2) control and user-plane disaggregation, either at the Core Network side through the adoption of SBA, or the RAN, through the adoption of architectures like O-RAN. In the O-RAN architecture, applications hosted on top at the edge of the network (*xApps* [7]) can retrieve statistics of the base station stack through standardized interfaces and analyze them for inferring features like network load, energy consumption, etc. Based on this inference, they can enforce policies regarding slice allocation and scheduling to ensure the smooth operation of the network. The inference relies on Machine Learning (ML) models, that can predict the future evolution of the monitored features/parameters, and thus apply proactively the target allocations. The O-RAN architecture can be further enhanced with the Network Data Analytics Function (NWDAF) which is standardized by 3GPP. NWDAF is a network-aware function that collects data from the 5G core and provides statistics to support network automation. These statistics can be employed by AI/ML models that run on RAN Intelligent Controllers (RIC) and can provide forecasting and optimization of Key Performance Indicators (KPIs) [8].

Leveraging Edge Intelligence, ML operations can be launched directly on the edge by taking advantage of several devices if needed in an entirely distributed manner, making use of pipelines. In this work, we design, develop, deploy and experimentally evaluate a service-aware network model for beyond 5G networks. We use a cloud-native network, with the entire stack (RAN and Core Network) being instantiated through the Kubernetes framework. We develop all the necessary extensions to support near-real-time ($\leq 10ms$) low-level monitoring of the traffic exchanged over the network. On top, and towards enabling accurate decisions for the slice allocations in the network, we use a distributed Machine Learning (ML) model, able to classify in real-time the traffic exchanged from the different users of the network and infer the future connectivity needs that are needed from the applications. The needs are in turn transformed into slice-allocation decisions for the 5G network. Our ML models have been developed in a distributed lightweight manner, allowing different parts of the training process to be executed at/near the edge devices, where processing power is usually limited. By decomposing the main model into lighter components and making extended use of pipelines, we are able to instantiate the framework at the edge and affect the wireless network allocations directly from there, thus augmenting the network with edge-located Intelligence.

Our contributions are summarized as follows:

- To develop a real-time classification model, hosted on the operator side of the network, recognizing the different applications that run on top of the network.
- To infer the future load and patterns of traffic from the different traffic flows of the applications that are hosted on top of the network.
- To decide on the slice allocation that is enforced in the network, based on the foreseen needs of the applications.
- To determine the optimal approach for predicting the future demand, from a set of different supervised ML models.
- To evaluate the developed scheme under real-world settings, using real devices and realistic traffic scenarios in real-time.

The rest of the paper is organized as follows. Section 2 presents our motivation, based on a recent literature review. Section 3 presents our overall system architecture, detailing the different components and their intercommunication, as well as an evaluation of the different ML models that drive our final choices. In Section 4 we evaluate our contributions and present our findings. Finally, in Section 6 we conclude the paper and present some future directions.

2. Related Work

The disaggregation of the telecommunications stack has been identified as one of the key enablers for flexibility, and further innovations for the beyond 5G and future 6G networks. By taking advantage of the disaggregation and existing approaches for an end-to-end SBA, the telecom stack can be instantiated as cloud-native functions throughout the resource continuum, thus allowing network operators to take advantage/extend existing approaches for VNF management, tailored to network-specific characteristics. Several of the works in the relevant literature

focus on managing the deployed components as VNFs, divided mainly into the following categories: 1) Placement of the VNFs [9],[10], [11], [12], 2) load that they are receiving [13], [14], [15], and 3) scale of the functions [16], [17], [18].

The most outstanding effort reflecting these architectural approaches is the definition of the Open-RAN (O-RAN) specifications [7]. O-RAN standardizes the interfaces for interacting in real-time, near real-time, and non-real-time with different components of the RAN stack, enabling the network to re-configure dynamically, based on operator-defined policies. Opening up the programmability of the RAN has created several opportunities for the integration of Artificial Intelligence methods, which infer based on historical observations of metrics on the future resource usage, and appropriately manage the network services.

In the realm of RICs for telecommunication networks, several solutions, both open-source and proprietary, are available. FlexRAN [19] stands out as a flexible and programmable platform tailored for Software-Defined Radio Access Networks (SD-RAN) and is compatible with the open-source OpenAirInterface (OAI) platform. Its successor, FlexRIC [20], serves as a software development kit (SDK) designed for next-generation SD-RANs, allowing its customization in the functions that the user needs to perform on the RAN. On the proprietary front, Athena Orchestrator—O-RAN SMO & RIC [21] is an AI-driven platform optimized for energy-saving management in 5G-ORAN compatible private networks. Additionally, FlexSlice [22] introduces an innovative approach, presenting flexible control logic topologies—centralized, decentralized, and distributed—to refine the O-RAN architecture for reduced control loop latency.

Different methods of Machine Learning are employed for the prediction of different network metrics, depending on the metrics themselves and their fluctuation to incoming load. For example, in [23], authors present a conceptual model for 6G networks and show the use and role of ML techniques in each layer of the model. Different ML methods are examined for the different parts of the stack, including supervised and unsupervised learning and Reinforcement Learning (RL). Regarding supervised learning, they employed Deep Learning (DL) in a distributed manner with the use of Federated Learning (FL). The application of ML has opted in several works dealing with the characterization of traffic exchanged over the network. For instance, in [24], the authors classified the traffic according to application and bandwidth-related features. Furthermore, the networking systems can identify factors that affect the operation of the network (e.g. external traffic for DDoS attacks) and appropriately employ the respective mechanisms for reinforcing the operation of the network (e.g. firewall operation, slicing of traffic, etc.). For example, in [25], authors employ a federated ML approach that can be ideally realized in networking switches, towards detecting intrusions in the network by processing packets at the bit level and at line-speed. In [26] authors use a non-parametric approach for traffic classification, which can improve the classification performance effectively by incorporating correlated information into the classification process, using the nearest-neighbour approach. Their approach demonstrates significant performance benefits from both theoretical and empirical perspectives in the literature. Authors in [27] employ cluster analysis for the case of peer-to-peer networks that use dynamic port numbers for the communication between participating nodes. Their presented approach demonstrates how cluster analysis can be used to effectively identify groups of traffic that are similar using only transport layer statistics. Finally, surveys [28, 29, 30, 31] organize the different traffic classification techniques that have emerged in literature for analyzing traffic based on either their headers, or the payload, and whether it is encrypted or not.

Similarly, in [32] authors propose the adoption of ML for orchestrating different tasks of 5G and beyond networks, such as massive MIMO, heterogeneous network integration and spectrum access, energy harvesting, and others. In [33], authors introduce the concept of xApps, running on top of the O-RAN architecture. These are network management applications, that rely on statistics exposed from the stack at different levels. Based on the decision time, *xApps* can be running in near real-time or non-real-time fashion. In [34] Thantharate et al. propose the ECO6G model, leveraging a Machine Learning approach to forecast traffic load for improved energy efficiency and OPEX savings in B5G networks. This research demonstrates that ECO6G significantly outperforms traditional forecasting methods in energy savings, presenting a vital step towards sustainable and cost-effective network management.

Regarding the type of policies and enforced decisions, several works deal solely with allocating resources for slicing the 5G network. In [35], authors employ Federated Learning as a means of predicting the evolution of each KPI in a per-service manner. Subsequently, they allocate the slices in the network. In [36], similar functionality is suggested, using the FlexRAN controller for reactively enforcing decisions regarding the network operation. Nevertheless, truly online training and decision-making in such systems pose a significant challenge, as model training can consume slice resources. Authors in [37] propose their solution for combating such issues with an online end-to-end network slicing system, able to achieve minimal resource usage while satisfying slices' Slice Level Agreements (SLAs). In [38] the Probabilistic Intra-slice Resource Service Scheduling (PRSS) algorithm is introduced to optimize 5G network resource allocation. Designed in two stages—service throughput estimation via a multinomial probabilistic model and dynamic conditional resource estimation for new services. Its efficiency is demonstrated through analytical and simulation results, showcasing its capability to efficiently manage 5G network resources.

In this work, we developed a solution for enhancing the network operation with intelligence, based on the type of

Table 1: Comparison of state-of-the-art with our approach.

Works	Approach	Evaluation
[33]	Open RAN for 6G networks focusing on modular traffic steering implementations.	Highlighted modular approach and AI/ML benefits in simulations; model lifecycle not discussed.
[34]	A supervised ML approach for forecasting traffic load to evaluate energy efficiency and OPEX savings in B5G networks.	Centers on model development and validation using real-world 5G data, omitting live deployment details and lifecycle discussions.
[35]	Uses Federated Learning to predict service-oriented KPIs for 5G network slices, addressing privacy and scalability challenges.	Proven in simulations to enhance KPI accuracy, ensure privacy, and cut communication costs. Highlights gaps in model lifecycle and scalability discussions.
[36]	A RAN runtime slicing system for flexible reactive slice customization in 5G networks, utilizing a runtime SDK for agile control application development.	Prototype development demonstrated on OpenAirInterface and Mosaic5G platforms, focusing on system capabilities.
[37]	Online DRL for dynamic end-to-end network slicing, focusing on SLA satisfaction and resource optimization.	Surpassed rule-based and DRL methods in resource efficiency and SLA compliance in simulations. Omits new traffic adaptation, real-world validation, and lifecycle management.
[38]	Introduces PRSS for optimizing 5G network slicing with a two-stage probabilistic model for resource estimation.	Demonstrated efficiency through analytical and simulation results. Lacks details on deployment, handling new traffic patterns, and model lifecycle management.
[39]	Slices resource orchestration using ML techniques for dynamic slicing of PRBs, admission control, and resource management.	Showcased better prediction and efficiency in simulations against static and random slicing. Lacks real-world deployment details and model lifecycle.
This work	A fully cloud-native, service-aware real-time network slicing model leveraging ML for traffic classification, mobility forecasting, and utilizes MLOps for model lifecycle management with online and distributed training.	Validated in a real-world environment; showcased superior latency and throughput improvements. Emphasizes practical deployment with a focus on adaptability and continuous optimization through a robust MLOps framework.

services hosted over the top. By employing a service classifier, we were able to determine in *real-time* the type of application running on the top and decide on the allocation of slices over the network in almost real-time. Moreover, our research stands out by implementing a thorough MLOps strategy, contrary to numerous previous studies that deploy deep learning models on fixed datasets, neglecting the emergence of new data patterns and the ongoing management of the model. To clarify our pivotal contributions within Table 1, we present a suite of innovative advancements that distinguish our research from existing state-of-the-art solutions:

- Leveraged the OpenAirInterface platform (OAI) for the RAN and Core Network, running in a cloud-native disaggregated manner using micro-services.
- Utilized programmable attenuators connected to the RAN to simulate realistic mobility scenarios.
- Implemented a custom NWDAF, enriching the dataset with metrics (throughput, jitter, CQIs) for enhanced traffic analytics and mobility insights.
- Used supervised learning to forecast various features and evaluated the solution with 6 different neural networks.
- Introduced and evaluated an MLOps architecture that leverages cloud/edge computing in the resource continuum for Online and Distributed Training among cluster nodes.
- Evaluated the framework in a real-world setup with commercial UEs connecting to the network, generating realistic traffic patterns.

3. System Architecture

Our experimental setup consists of a cloud-native disaggregated 5G network fully deployed on the Kubernetes framework. This way, we take advantage of the multiple benefits provided by an application container orchestrator like Kubernetes, such as the management and monitoring of resources and dynamic scaling of the 5G VNFs. The 5G network is enriched by a novel distributed AI/ML unit for continuous distributed training-prediction and slicing. Fig. 1 summarizes the framework's architecture, showing the deployment of the service-based 5G network, the introduced distributed AI/ML unit, and the internet applications that the end-users interact with. We deploy the framework in the NITOS testbed [40], a remotely accessible facility located at the University of Thessaly, Greece. NITOS testbed provides Software Defined Radios (SDRs), User Equipment (UE) terminals, and programmable attenuators. All these devices are utilized to develop our solution in a real-world environment. Below,

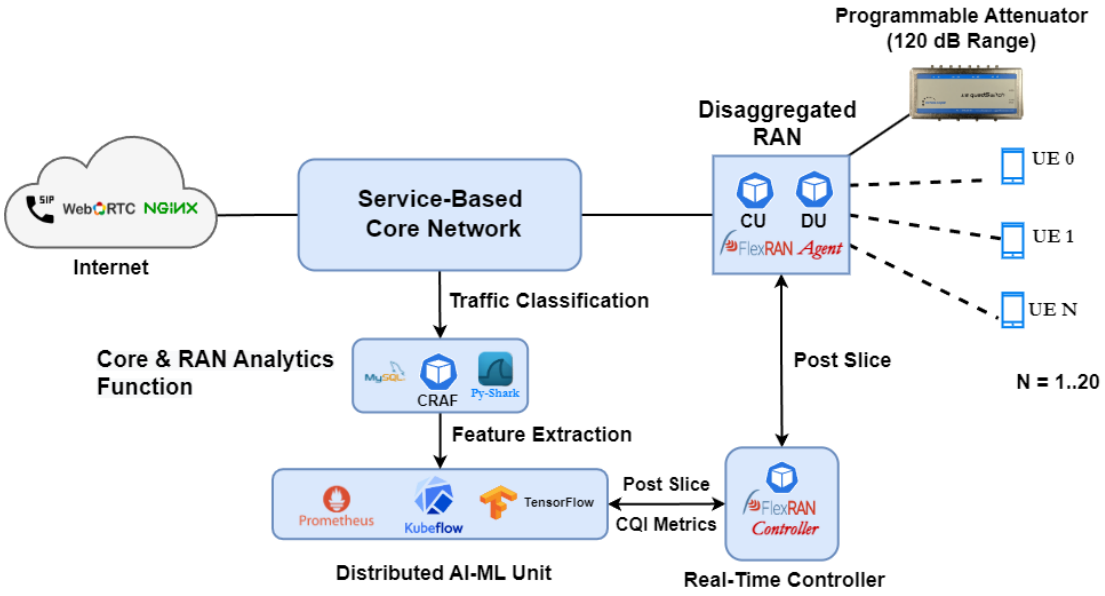


Figure 1: Experimental Setup - The deployment of Cloud Native-AI 5G Network on Kubernetes.

we list the essential elements of our AI network slicing solution that enables provisioning high QoS and continuously user-perceived high QoE.

3.1. Management and deployment of the network functions

Our telecom network follows a serviced-based architecture which consists of containerized network functions. The containerized deployment relies on the open-source OpenAirInterface platform. We specifically leverage the LTE implementation of the OAI platform, opting for its stability and mature RAN slicing support for multiple User Equipment (UEs), a feature not yet fully developed in the current OAI 5G NR implementation. Despite this, our solution seamlessly integrates with 5G architecture, requiring minimal adjustments to the overall framework. For instance, substituting the LTE Evolved Packet Core (EPC) with 5G core network components (HSS/UDM, MME/AMF, SPGW-U/UPF, SPGW-C/SMF) and transitioning from a disaggregated eNB to a disaggregated gNB can be achieved effortlessly. It's worth noting that our approach to the LTE Evolved Packet Core (EPC) involves the use of Control and User-Plane Separation (CUPS), allowing each component to operate in isolation. Our work focuses on RAN-level allocations, utilizing interfaces envisaged for 6G network operation, such as the O-RAN E2. Notably, our solution remains independent of dedicated slicing components from the 5G architecture, like the Network Slice Selection Function (NSSF). The key distinction with the 5G RAN lies in the absence of full slicing support, with the primary difference being the data rate rather than core functionalities. For the experimental evaluation of our architecture, we created a cluster of three NITOS nodes as Kubernetes workers, while the control-plane node was running on a separate VM. Below, we analyze our cloud-native approach for the deployment of the network functions down from the core network, up to the end-user.

3.1.1. Service-Based Core Network

The core network architecture follows control and user-plane separation (CUPS). Consequently, each function runs as a separate pod/container providing: a Cassandra database that holds the subscriptions, the Home Subscriber Service (HSS), the Mobility Management Entity (MME), the control plane Service/PDN Gateway (SPGW-C), and the respective user plane service (SPGW-U). Since there's not yet an open-source implementation of the NWDAF we developed a customized function named Core RAN Analytics Function (CRAF). CRAF plays the same role as NWDAF in our architecture. It collects traffic statistics from application interactions and KPI network metrics such as Throughput, Jitter, and the CQI. After the collection of the data, CRAF stores them in a database. Then, our AI/ML framework performs feature extraction and preprocesses the data for the model training.

The fact that the individual core network components run separately as micro-services allows us to easily monitor their status and their consumption in terms of memory, CPU, and bandwidth. The deployment of the core network is distributed to all Kubernetes workers ensuring the load balancing between them. The connectivity between the containerized core network and the Radio Access Network is realized by the Multus Container Network Interface (CNI). Multus CNI allows us to provide multiple interfaces to pods and create static network configurations for easy reproducibility of the experiments.

3.1.2. Disaggregated RAN

The containerized Radio Access Network (RAN) follows a disaggregated architecture including the CU and DU (Central & Distributed Unit) components. This distributed scheme implements the functional split of the base station. Specifically, the split takes place in the layer 2 OSI stack, between Packet Data Convergence Protocol (PDCP) and Radio Link Control (RLC) layers. The CU integrates the upper layers, while the DU integrates the lower layers (from the RLC and below). The communication between CU and DU is based on the F1 Application via the F1 interface. The CU container can be deployed in any of the Kubernetes nodes from our cluster, contrary to the DU pod that needs to be deployed on a specific node equipped with the appropriate SDR front device. In the SDR device, a programmable attenuator is connected, with which we attenuate the signal of the RF device, in order to create realistic mobility scenarios.

To obtain RAN statistics such as CQI and to create network slices on demand, we utilize the FlexRAN network controller. FlexRAN provides flexible and efficient resource allocation and by this time of writing, is the most stable open-source solution for RAN slicing. We connect the FlexRAN controller to the RAN via the FlexRAN agent running on the CU/DU side. FlexRAN is also connected to the CRAF and AI/ML unit ambiguously for the transmission of the RAN statistics and to the establishment of the slicing policies.

3.1.3. End-Users & Internet Applications

To evaluate the network connectivity and collect traffic data, we connected 3 UEs to the network interacting with 3 containerized applications on the internet. The mobile equipment includes commercial UEs by utilizing LTE dongles. The applications include a video streaming service, a VoIP application, and an Nginx web server. The reason for choosing these services is to classify their network needs into data-hungry applications such as video streaming, medium data-rate applications such as VoIP, and low data-rate applications such as simple web-server. The video streaming service streams video capture devices by utilizing the webRTC protocol as it provides real-time communication over the web. The VoIP service is an application called SiPp that employs Session Initiation Protocol (SIP) for VoIP packet transferring. The Nginx web server is employed for the generation of HTTP requests. All services are containerized and deployed onto the same Kubernetes cluster. This allows us, to deploy them among the SPGW pods on the Node with the SDR device to provide an edge computing approach. Finally, the traffic can be captured and fed to the CRAF, directly from the SGi interface of the data-plane network.

3.2. Application-aware AI/ML Unit

Developing an efficient AI/ML unit, aware of the network conditions that coordinates the resources optimally requires considering a lot of parameters. Our approach captures a large number of features, essential for the slicing decision, including the applications used by every UE, the Throughput, and the Channel Quality, among many others. Noticeably, the model receives an input window of multiple time slots, with these features, which represent the network traffic exchanges between the UEs and the applications in the near past. Thus, the model identifies the pattern in the traffic and predicts future values. Our goal is to come up with a robust unit that analyzes the overall network conditions thoroughly and employs a superior slicing allocation algorithm, leading, this way, to a network performance peak. Below, we provide information on the whole procedure of choosing the proper features, designing an effective traffic classification scheme, creating real-world network traffic scenarios in the experimental environment, collecting data, training multiple models, and developing a novel near real-time slicing allocation scheme.

3.2.1. Feature Selection

Designing a powerful AI/ML model, aware of the plethora of components in a network architecture requires a cautious feature selection. Thus, we pick many features to capture the largest possible variance that explains the pattern underlying the traffic exchanges between the UEs and the applications (apps). Precisely, our features' list consists of the *Applications*, the *Throughput*, the *CQI*, the *Jitter* and the allocated *Slices*, for every UE of the network. First, the *Application/Service* is a principal component of a service-aware implementation capturing which specific service is used by every UE. This feature indicates the service's type, demand, and significance. Importantly, for every UE, we keep one feature for every application provided by the network; in our case, there are 3 app-features (*WebRTC*, *SIPp* and *Nginx*). Further, the experienced service *Throughput* provides essential information about the bandwidth of the UE-App link. Another vital feature is the *CQI* that represents the LTE channel quality, which demonstrates the quality of the UE connection; indicating a great or poor connection. Moreover, the *Jitter* monitoring per UE depicts the timing delays between the UE's packets, while the *Slices* show the allocated resource blocks of every UE.

3.2.2. Traffic Classification

For traffic classification, we divide the timeline of every experiment into multiple time slots of a fixed length, in which we gather the desired network information with the aforementioned features.Importantly, the information in every time slot is organized in a specific structure. We divide every time slot into multiple UE categories as shown in Fig. 2. This way, the information for every UE is gathered in one category. In our case, there are 6 different features for every UE category, namely *WebRTC, Sipp, Nginx, CQI, Jitter* and *Slice*. The first three features represent the network *Services* that the UE is able to use. Noticeably, their values represent the *Throughput* of the

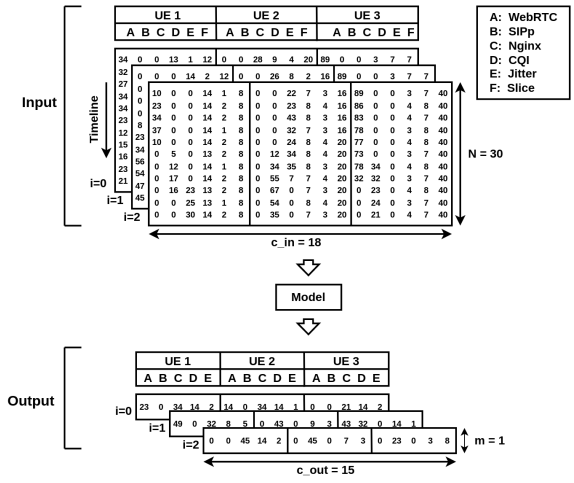


Figure 2: Traffic Classification & Sliding Window Approach

specific UE with the specific service. For instance, a value of 10 in the *WebRTC* feature in the first category (UE 1) is translated as 10 Mbps network traffic on the UE 1 using the *WebRTC* service. The remaining features of every category, namely *CQI, Jitter* and *Slice* provide additional information on the quality of the UE connection as well as its allocated resources. As a result, we end up with a number of columns that is proportional to the number of UEs multiplied by the number of features per category; in our case, 3 UEs multiplied by 6 features equals 18 total columns (real features for the model) for every time slot. This is illustrated in Fig. 2, where every column of the tables is a feature and every row is a time slot.

This way, we organize the monitored network traffic into a useful structure to be used by a model. Precisely, the time slot length is configured to the desired number, for instance, 100 ms. Subsequently, during every slot, we gather all the received packets and extract the essential information. Firstly, we read the packets' IP/Transport protocols to classify them to the appropriate UE-App combination. Then, we count the total number of bytes of all packets received during the time slot for every UE-App link to calculate the Throughput. This way, we classify the captured traffic during a time slot to the appropriate columns. Next, we compute the mean Jitter value between the total packets of every UE in the time slot. On top of that, a CQI value per UE is requested from the FlexRAN Agent existing in the LTE DU, and finally, the currently allocated UE slices are recorded as well. For a better understanding, let's focus on Fig. 2 in the first row of the third input window ($i = 2$). The first 6 values corresponding to the category of the UE 1 are:

$$(A, B, C, D, E, F) = (10, 0, 0, 14, 1, 8)$$

Interpreting this category, we understand that the UE 1 has 10 Mbps network traffic only with the WebRTC service, an LTE CQI of 14, 1 ms average Jitter, and allocates a slice of only 8% of the overall network resources.

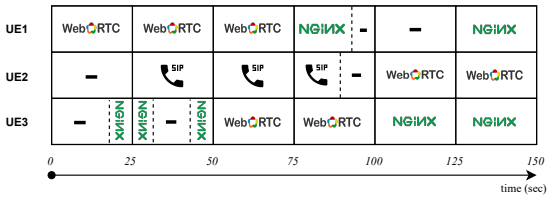


Figure 3: Users' Network Traffic Baseline Scenarios depicting network traffic at a specific time interval during the day.

3.2.3. Real-world Traffic Scenarios

We emulate realistic network behavior in an office by developing multiple network traffic scenarios. Our goal is to emulate inside our experimental infrastructure the network patterns observed in an office on a specific time interval of a usual day. We aim at specific time intervals and not the whole day since our resources are limited. Most users in an office are expected to have a basic pattern in their behavior. For example, one user might mainly utilize video streaming platforms, whereas another one is constantly on calls with clients. Thus, the AI/ML unit captures this pattern and enhances users' overall experience by sharing the network resources on demand. As a first step towards emulating this office behavior, we create some baseline traffic scenarios for every UE in our network (one bash script per UE specifying a particular behavior) as shown in Fig. 3. These scenarios are based on real network patterns observed at a specific time interval during the day (early morning from 10:00 AM to 11:00 AM) on users in our office facilities in Volos, Greece. However, we redesign them to be small with a duration of approx. 150-160 seconds to facilitate the whole experimental procedure on the testbed. This way, we create the basic pattern that is observed in our office at that specific time interval. However, this is not the exact behavior every day since it will slightly change from one day to the other even if the underlying pattern is the same. For example, the employee who works mainly on the phone will not make the same number of calls or calls of the same duration every day, but he/she will mainly work on the phone with clients. To emulate these slight variations in the UE behaviors from day to day, we employ data augmentation techniques. Specifically, based on the baseline scenarios, we add Additive White Gaussian Noise (AWGN) in the number, sequence, starting time, and duration of the utilized applications by a UE to represent the differences from one day to the other. For instance, the UE 3 in Fig 3 uses the WebRTC app one time starting at 50 secs for a duration of 50 secs. It also uses the NGINX app three times in total each starting at about 20, 45, and 100 secs for a duration of 10, 5, and 50 secs respectively. At first, AWGN from the standard normal distribution with a mean of 0 and a standard deviation (sd) of 1 is added to the number of times that an app is used. Regarding UE 3, this means that the number of times that the WebRTC and NGINX are utilized will either not change or increase/decrease up to a maximum

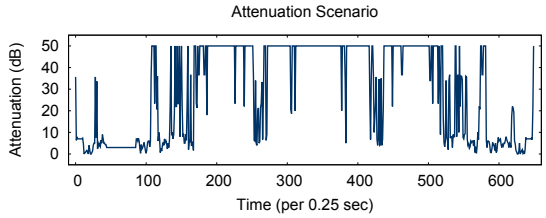


Figure 4: Attenuation Scenario emulating UE mobility in office.

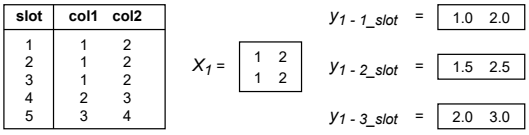


Figure 5: Example of sliding-window scheme.

of 3 times (3 standard deviations from the mean). Then according to the new numbers we add the new apps or delete the unnecessary ones randomly. Subsequently, we use the same distribution to choose randomly several apps (up to three) and change their position in the timeline. Then, AWGN from a different distribution (mean of 0, sd of 10) is added to change the starting time of each app up to a maximum of 30 secs (3 sd from mean). After that, AWGN from the same distribution is inserted to change the duration of each app increasing or decreasing it by a margin (up to 30 secs - 3 sd from mean). At every step, we adjust accordingly the position of the apps in order to avoid interference.

Moreover, several scenarios are reversed to augment the dataset further and a lot of them are slightly cropped for efficient training. Further but minor noise is inserted when we collect the data from the testbed due to hardware imperfections. Thus, we create a plethora of network traffic scenarios for every UE that inherit the baseline pattern but are slightly modified capturing a large spectrum of the office's real traffic at that specific time interval. Hence, there is a large variance to build robust AI methods, capable of generalizing, not over-fitting, and being resilient to noise and fluctuations.

3.2.4. UE Mobility Emulation

In a real network, the quality of the UE connection varies according to the geographical location of the UE. Specifically, in areas with good LTE coverage the CQI that depicts the LTE channel quality, is high, in contrast with areas where there is poor LTE coverage (low CQI). In order to emulate this behavior in our experiment we use programmable attenuators installed on the outputs of the USRP, as presented in Fig. 1. Specifically, by modifying the attenuation of the USRP radios, we can emulate transitions from low to high CQI values and vice versa. The attenuation is inversely proportional to the CQI (high attenuation causes low CQI and the opposite). Importantly, we possess attenuation scenarios from real commercial networks in Volos, Greece. Specifically, these attenuation scenarios emulate cars traveling a specific city route with velocities that vary from 40 to 60 km/h with the road's limit being 50 km/h. These car scenarios were used to collect 182500 CQI data from 73 cars capturing a large spectrum of the route's traffic. The CQI data are publicly available [41]. We decide to utilize the same attenuation scenarios to emulate mobility to the office users since it is a similar problem (users moving in a specific geographical area) and moreover, because it is a dataset with a large variance that could lead to efficient training and generalization of the models. Fig. 4 depicts an attenuation pattern used, where at the beginning of the experiment the attenuation is low (high CQI). Following that, the attenuation rises substantially (low CQI), while at the end of the experiment, the attenuation returns to low levels (high CQI).

3.2.5. Data Collection

To collect a lot of training examples for our model, we execute all the scenarios in the testbed. In specific, we pick at random one of the traffic scenarios (office users' pattern) and one of the attenuation scenarios (mobility pattern) and execute them concurrently. This way, we assign a different combination of office traffic and mobility patterns to each experiment. Meanwhile, by employing the traffic classification scheme with a time-slot duration of 1 second, the network traffic is appropriately classified and subsequently stored in the database. This is done for 300 experiments (each lasts approx 150-160 seconds) creating, as a result, a massive dataset with 48600 rows and 18 columns. This dataset is also publicly available [42].

3.2.6. Pre-processing

Before feeding the data into the models, we need to preprocess them appropriately. First, we normalize the whole dataset adjusting all the columns in one common range between 0 and 1. This way, we avoid scale imbalances strengthening the model's training efficiency. Subsequently, Fig. 2 illustrates clearly our pre-processing technique. In specific, we utilize a sliding-window approach which creates a 2D input window (X_i) of fixed shape ([N time slots, c_{in} features]) and slides it by one-time slot over the whole dataset to create multiple samples ($i = 0, i = 1, i = 2$). Meanwhile, for every X_i sample, the algorithm captures a second 1D output window (y_i) with shape [1, c_{out} features], which depicts the data that we want to predict (labels) The data of every 1D window (y_i) are located immediately after that of the 2D window (X_i) in the dataset representing the future. Noticeably, the values of each y_i could be that of only one-time slot (the following of the X_i) or the mean values of an arbitrary number of time slots following the X_i. For example, we provide a dataset with shape [5,2] in Fig. 5 Given that we want to pick X_i windows with a length of 2-time slots, the first input sample (X_1) would be the first two

rows. For the corresponding prediction-output window y_i there are a lot of choices depending on the number of future time slots that we want to predict. For instance, to predict one future time slot, the y_1 would be the third row. On the other side, to predict multiple future time slots, one efficient solution is to obtain the average values of their columns. Fig. 5 demonstrates examples for predictions of 1, 2, and 3 future time slots: For the following X_i, y_i samples, we slide by one-time slot and apply the same procedure until we reach the end of the dataset. In our case, as shown in Fig. 2, after extensive experimentation we conclude on calculating X_i windows with shape [30,18] and y_i vectors of shape [1,15] predicting the average values of five future time slots. The general rule for finding the optimal window shapes is that the X_i windows should be sufficiently large to capture the pattern in the near past but small enough to boost model training and avoid the exploding/vanishing gradient problem when Recurrent Neural Networks (RNNs) are used. Regarding the number of future time slots for prediction, it is generally good to employ multiple future time-slots to smooth possible fluctuations, but not too many of them so as to present an accurate figure of the near future. Using this technique, we structure the data in X_i samples of shape $[48566, 30, 18]$ and y_i samples of shape $[48566, 15]$.

3.2.7. Neural Network Models

This work focuses on supervised learning approaches and specifically, on evaluating various deep learning methods. We focus on neural networks as they are generally more robust at handling huge datasets and more resilient to noise compared to statistical and tree-based methods. Our goal is to design a robust Neural Network (NN) that converges on the pattern fast and accurately in order to be used for real-time forecasting implementation. Hence, we explore many different NN structures and finally conclude on some of the most promising ones and provide their specifications in Table 2.

Firstly, we choose a Feed-forward NN (FNN) due to its simplicity by just moving the information forward from the input to the hidden and to the output layers resulting in faster training. Subsequently, we move to more sophisticated architectures, the Recurrent NNs (RNNs), which employ memory components and are widely utilized in Time Series Forecasting (TSF). Precisely, Long short-term Memory (LSTM) NN are very robust at dealing with the vanishing/exploding gradients issue using three gates (input, output, and forget gates) and thus, they often outcompete simpler RNNs. Following that, we extend the simple LSTM by inserting a Bidirectional layer (Bi-LSTMs). This way, the model analyzes both the original sequences and their reversed versions, obtaining information from the past and also the future, usually resulting in enhanced forecasting performance. After that, we analyze Gated Recurrent Units (GRUs) NNs, another widely used RNN, that achieves similar predictive performance with LSTMs. In fact, GRU is equipped with fewer gates (reset and update gates) and hence, requires fewer training parameters leading to faster training. Then, we build a Convolutional NN (CNN) that is powerful at efficiently extracting features, dealing with noise, reducing the dimensions, and calculating non-linear functions in data by employing kernel filters, pooling layers, and fully-connected layers. Consequently, they often result in more accurate and fast training. Further, we experiment with a hybrid CNN-LSTM that obtains the best from both worlds by forming an Encoder-Decoder architecture. In specific, the CNN part implements feature extraction, noise, and dimensionality reduction and subsequently passes the processed information to the LSTM, which captures the pattern in data using memory components. This way, the result is a prominent model with remarkable predictive and training performance.

	Layers	**Hidden Layers**	**Epochs**
GRU	2 GRU + Dense	25 units per layer	61
LSTM	2 LSTM + Dense	25 units per layer	97
Bi-LSTM	2 Bi-LSTM + Dense	25 units per layer	56
FNN	2 Dense + output Dense	25 units per layer	568
CNN	Conv1D + MaxPooling1D + Flatten + Dense + Dense (output)	*filters=64, kernel size=2, pool size=2, 50 Dense units*	264
CNN-LSTM	Conv1D + MaxPooling1D + Flatten + RepeatVector + 2 LSTM + Dense (output)	*filters=64, kernel size=3, pool size=2, repetition factor=1, 25 units per LSTM layer*	24

Table 2: Neural Networks Configuration

3.2.8. Slicing Allocation Mechanism

The slicing allocation algorithm is designed to provide the network resources on demand and fairly to maximize the Quality of Experience (QoE) of the UEs. To achieve that we share the available network resource blocks based on a mathematical formula that consists of many criteria obtained from the model predictions. Precisely, the type of the application (C_1), the total Throughput of the UE (C_2), the CQI (C_3), and the Jitter (C_4):

$$Slice(\%) = \sum_{i=1}^{4} (w_i C_i) + w_0, \tag{1}$$

where w_1, w_2, w_3, w_4 are the weights of every criterion indicating its importance and w_0 is a constant term representing the minimum value of the slice.

Each criterion (C_i) is assigned a priority value (0, 1, or 2), signifying low, medium, or high importance, respectively. For example:

- For UE application (C_1), WebRTC is given the highest priority (2), followed by SIPp and Nginx with priorities 1 and 0 correspondingly.
- Throughput (C_2) is classified as high demand (2) for values above 0.4 Mbps, medium demand (1) for values

between 0.2 and 0.4 Mbps, and minor demand (0) for values below 0.2 Mbps.

- CQI values (C_3) falling between 0 to 9 are high priority (2), 9 to 11 are medium priority (1), and above 11 are low priority (0).
- Jitter values (C_4) of more than 10 ms are crucial (2), 5 to 10 ms are medium priority (1), and less than 5 ms are low priority (0).

After experimenting with various slice configurations, we determined that in our experimental setup, maintaining a minimum slice value of 8% is crucial to keep a User Equipment (UE) connected to the network. Any value below this threshold results in UE disconnection, prompting us to establish 8% as the designated minimum slice value (w_0). Additionally, we observed that UEs achieve their optimal performance when allocated a slice of 40%. Beyond this value, there is no discernible increase in connection efficiency. Consequently, we selected 40% as the maximum slice value. This maximum value is determined when all criteria in Eq. 1 have the highest priority:

$$40 = w_1 \times 2 + w_2 \times 2 + w_3 \times 2 + w_4 \times 2 + 8$$

In our study, we assigned equal importance to each criterion, reflected in identical weight values for w_1, w_2, w_3, w_4, all calculated as 4. Consequently, the slicing equation simplifies to:

$$Slice(\%) = 4 \sum_{i=1}^{4} C_i + 8 \quad (2)$$

Various strategies can be implemented by assigning different weights to individual criteria based on specific objectives. For instance, prioritizing Ultra-reliable Low Latency Communications (URLLC) would involve assigning a higher weight to the *Jitter* criterion (C_4). This adjustment enhances the slice allocation sensitivity to Jitter, ensuring that more resources are allocated to UEs experiencing Jitter fluctuations. Alternatively, assigning greater weight to *Throughput* (C_2) could strengthen support for Enhanced Mobile Broadband (eMBB), while an emphasis on the weight of *CQI* (C_3) would focus on maintaining a stable, high-quality connection. Similarly, allocating more weight to *Application* (C_1) would result in additional resources based on the application type rather than the quality of the connection.

In our case, we choose an equal weight to all criteria to evaluate the algorithm's general efficiency as a first step. Future works will focus on specific use cases. Table 3 adduces examples of the slicing allocation algorithm for further understanding. For instance, the forecasting regarding the UE 1 indicates that the Nginx app will be utilized with 0.1 Mbps Throughput, a CQI of 14, and a Jitter of 2 ms. All these values correspond to the lowest priority (0) of each criterion (C_i) and thus, the calculated slice is the lowest, 8%. At UE 2 and 3, all criteria have medium and maximum priority leading to a slice of 24% and 40% respectively.

Forecasting	**UE 1**	**UE 2**	**UE 3**
Application	Nginx	SIPp	WebRTC
Throughput (Mbps)	0.1	0.3	2
CQI	14	10	6
Jitter (ms)	2	8	12
Criterion	**UE 1**	**UE 2**	**UE 3**
C_1	0	1	2
C_2	0	1	2
C_3	0	1	2
C_4	0	1	2
Slice(%)	**8**	**24**	**40**

Table 3: Examples of UE slices assigning the priorities to each criterion (C_i) based on forecasting.

When the total slices of the UEs are calculated more than 100%, we subtract an equal proportion of every slice. Overall, the UE receives the appropriate amount of resources depending on the network conditions without under or over-provisioning. In general, this scheme could be adapted to individual preferences. First, further criteria could be added or some of them could be excluded. Secondly, the weights could be adjusted on the individual preferences to target specific use cases. Additionally, the minimum and maximum values of the UE slice could be modified. Finally, this Eq. is a linear relationship between the criteria and the slice, and thus in the future, it could be replaced by a non-linear function calculated by an ML model.

3.3. MLOps AI-ML Unit Architecture

To ensure that our model adjusts to the training data's gradual drift, we employ an online/distributed training architecture realized by a Kubeflow pipeline. Kubeflow is an open-source AI/ML toolkit that utilizes the power of Kubernetes to run ML jobs and supports the entire lifecycle of ML applications. In Kubeflow, a pipeline is a description of an ML workflow that includes containerized components, each of which represents a single step in the process. Each element is managed as a microservice, with all the expected declarative definitions (YAML manifests). This, enables them to be quickly deployed and scaled out as required. By employing Kubeflow [43] pipelines we can easily orchestrate, scale, and automate our AI solution. This MLOps - Distributed Architecture is presented in Fig. 6. First, CRAF monitors all the traffic from the SGi interface by utilizing PyShark [44]. In order for CRAF to collect the traffic in real-time, we use the *LiveCapture* class of PyShark. CRAF also obtains all the CQI values in real-time, via HTTP requests from the FlexRAN controller. Then, after applying network filters to the traffic (IPs/Ports), it classifies the interactions per UE and application and calculates traffic analytics such us Throughput and Jitter. To avoid big data over time,

CRAF only keeps the summary of each packet such as the UE, the Application, the Length, the Jitter, and the CQI value that each UE experiences. Subsequently, this data is stored on a database running on a MySQL server that is backed with NFS persistent storage via PersistentVolume, providing consistency and availability of data between Kubernetes Nodes. Next, the Kubeflow pipeline takes place, as the first step: the Data Parser extracts the features from the database and creates a new dataset. Then, the next pipeline component, the Data Preprocessing applies the sliding window approach to the data and stores them in a multi-dimensional array. Afterward, this newly shaped array is passed to the last step of the pipeline, the Training component. The construction, and the training of the model, are implemented in this final step. After the train finishes the new model is saved on the NFS as an HDF5 file via the mounted Persistent Volume that is attached to the container. This way, the Predictor Service can obtain and utilize the updated model as it has access to distributed storage as well. As a result, the Predictor pod can make live predictions for near future traffic with higher accuracy, as the model is trained with the data with the most recent interactions and the latest network conditions.

To calculate the overhead of our solution we rely on the Eq. 3. It is the total time that is needed per slice allocation. All the metrics are measured with the help of timeit python module. The first metric, t_{CRAF}, is the total time for CRAF to obtain traffic and RAN analytics in one iteration. We measured that t_{CRAF} is almost real-time: 1-6 ms. The time needed for slice allocation t_{apply} is also in the same real-time range. This seems reasonable since CRAF employs PyShark for live packet capturing and FlexRAN for RAN statistics, which operates in real-time. Also, the overhead of each prediction (t_{pred}) is 1.6 ms. Finally, the catalytic factor of Eq. plays the time slot per X_i observation described by t_{slot}. We choose to observe X_i every 1 sec to get a better picture and capture the patterns. However, the time slot is a hyperparameter that can be changed. The smaller it becomes, the faster the slice allocations, with the only tradeoff being the efficiency of the predictions.

$$Slice_{time} = t_{CRAF} + t_{slot} + t_{pred} + t_{apply} \quad (3)$$

The pipeline can be triggered by the Predictor Service periodically with a timer or each time the predicted data is less accurate than a predefined threshold. This can indicate that the new data that is fitted into the model has different traffic patterns than the data that the model has been trained with. In that case, an algorithm 1 is suggested. As long as the accuracy (*R-squared*) of the forecasts is high, the slice decisions defined by the slicing Eq. 1 can be determined by the predictions. Otherwise, if the accuracy is lower than the accuracy threshold, then the slice decisions will be reactively determined by the slicing Eq. directly. The tradeoff in this approach is the fact that in the middle of the train of the updated model, we might lose some important interactions of the users with the applications as well as the new patterns of the network conditions (e.g. low CQI values). However, based on our experiments this algorithm can converge on new traffic patterns over time as the accuracy remains at constant-high percentages from one point onwards.

Algorithm 1: Online Train/Predict [Predictor Service]

```
Function model_select(pipeline_name, accuracy_threshold):
    train_flag = 0;
    while True do
        traffic_data = get_traffic_data();
        accuracy = get_accuracy_of_predictions();
        if train_flag == 0 then
            if accuracy > accuracy_threshold then
                model = get_model();
                yhat = model.predict(traffic_data)
                store_predictions(yhat)
                slice_perc = slice_desicion(yhat)
            else
                slice_perc = sclice_desicion(traffic_data)
                train_flag = 1
                trigger_pipeline(pipeline_name)
            end
        end
        else
            slice_perc = slice_decision(traffic_data)
            if status.pipeline() == True then
                train_flag = 0
                accuracy = MAX_ACCURACY
        end
    end
End Function
```

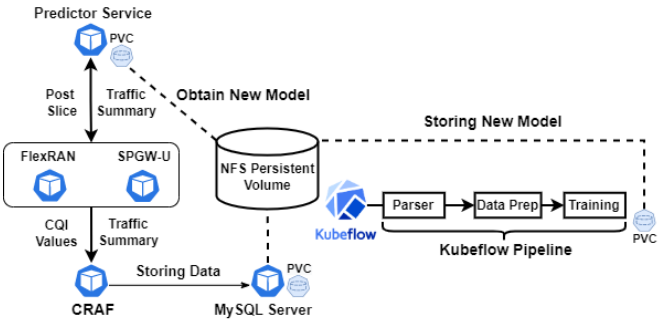


Figure 6: MLOps Training Architecture.

Towards aiming to reduce training time as much as possible and to distribute the training load evenly in the Kubernetes cluster, we enrich our architecture by employing Distributed training using Kubeflow's TensorFlow operator. With the TensorFlow operator, we can run distributed TensorFlow jobs (TF jobs) in our Kubernetes cluster as illustrated in Fig. 7. A distributed TF job is the collection of the following processes:

- Chief: Is responsible for orchestrating the training process
- PS: Parameter Servers provide a distributed data storage for the model parameters and perform gradient updates.

- Worker: The workers do the actual work of training the model.

Kubeflow handles the above processes by passing the Kubernetes cluster configuration as an environment variable to the TF jobs. We only define distributed strategies into our code for synchronous training based on the all-reduce algorithm or for asynchronous training via parameter server. In our experiments, we choose Multi-Worker with All-Reduce strategy and RING communication as it supports synchronous training, without suffering from bottleneck communications, contrary to the parameter server asynchronous training [45]. The distribution scheme can be further extended by describing the training job with a custom YAML file that references the TFJob Custom Resource Definition (CRD). In this way, we can scale our training process into multiple pods that will train the model in a distributed fashion taking advantage of the total resources of the cluster.

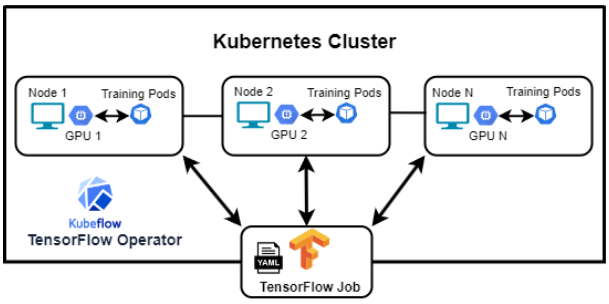


Figure 7: Distributed Training

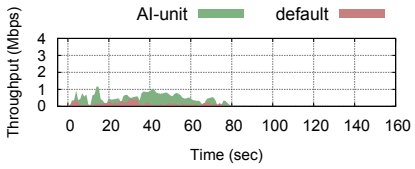
4. Evaluation

4.1. Model Comparison

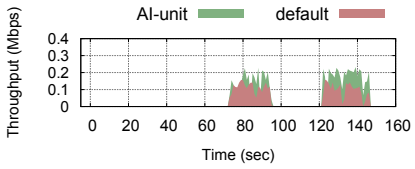
The models' offline training and evaluation are taking place on Google Colab where non-subscription TPUs are used. The concluded/optimal model structures are analyzed in Table 2. To evaluate them, we employ Time-Series Cross Validation (CV), a technique similar to K-fold CV but designed to respect the time sequence. We split the pre-processed data (48566 X_i, y_i samples) into several folds of equal size (500 samples) and create two sets; the training and the testing one. At first, we initialize the training set with multiple serial folds following the timeline (32000 samples - data of about 200 experiments). On every iteration (i), the model is trained on the training set and uses the next fold on the timeline as a testing set to calculate the generalization error on unseen data. In the following iteration, the training set is increased by one fold following the timeline, and the next one is used for a new evaluation. In the end, the mean of all testing errors (data from about. 100 experiments) is calculated as the overall generalization error. As a second step, we pick each model and integrate it into our experimental topology to evaluate its predictive performance in realistic circumstances on our Testbed. The time-series CV and Testbed's experimental evaluations are shown in Fig. 11

As evaluation metrics, we employ the Mean Absolute Error (MAE) and the Coefficient of Determination (R^2). MAE finds the mean absolute error between the predictions ($\hat{y}_i$) and the labels. It is scale-dependent helping us understand the forecasting error when studied together with the data range and distribution. We calculate separate MAE values for the predicted UE-App Throughput, UE Jitter, and UE CQI both for the Time-Series CV and the Testbed's experimental evaluation, as shown in Fig. 11. Regarding Throughput, we observe a range of 0-800 kilobits per second (Kbps) with poor slicing and a range of 0-4 megabits per second (Mbps) with maximum slicing when the utilized application is the WebRTC. On the other hand, when Nginx and SIPp are used, the range is between 0-300 Kbps. Generally, the observed Throughput range in our experiments is between 0-4 Mbps. Regarding Jitter, the observed range is between 0-70 milliseconds (ms) depending on the link quality, slice, and application. Moreover, CQI ranges from 0 to 15. Further, we employ the R^2 metric, which calculates the proportion of total variation of outcomes explained by the model. It is more intuitively informative (percentage value) without the need to consider the data ranges.

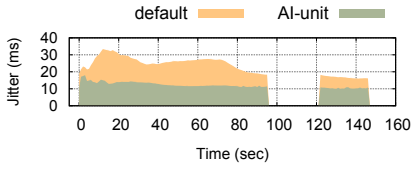
In Fig. 11 all the models identify the pattern in data efficiently. In specific, in Fig. 11a the models have time-series CV Throughput MAE values that range from 5.04 to 5.82 kbps, while the respective ones on the Testbed range from 2.07 to 3 kbps. These error values are negligible when compared with the throughput range, which is 0-4 Mbps. Additionally, the NNs predict accurately the experimental Jitters (Fig. 11b) reaching MAE values at just around 0.25 ms; very minor when studied with the Jitter range of [0-70 ms]. Moreover, regarding the CQI in 11c, the models achieve exceptionally low testing error with an average of 0.42 MAE considering that CQI ranges from 0 to 15. Moreover, the evaluation utilizing the R^2 metric on the time-series CV and on the experiments on the Testbed are shown in Table 4. Overall, the NNs achieve substantial performances, with each model being slightly better in forecasting different features. Importantly, there is a great discrepancy in their training time, as shown in Fig. 11d. The CNN-LSTM identifies quickly the patterns requiring only 4 minutes, while the remaining models demand from 26 to 76 minutes. The key enabler of CNN-LSTM's training efficiency is its convolutional (CNN) part. In specific, the CNN performs optimally feature extraction, noise, and dimensionality reduction. As a result, the LSTM part finds smaller and better-structured sequences being able to converge on the patterns in a faster way. Thus, we pick this algorithm and integrate it into the AI/ML unit as it combines high predictive accuracy with extremely low training time, being the most appropriate choice for our implementation.



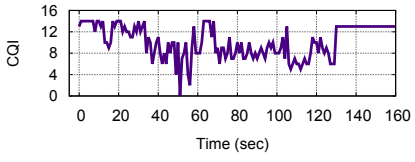
(a) UE 1 interaction with WebRTC service



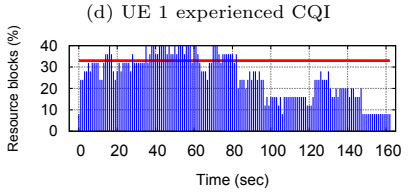
(b) UE 1 interaction with Nginx service



(c) UE 1 experienced Jitter

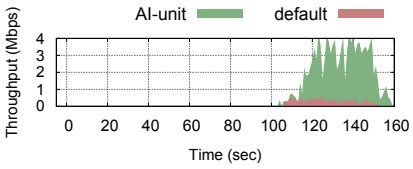


(d) UE 1 experienced CQI

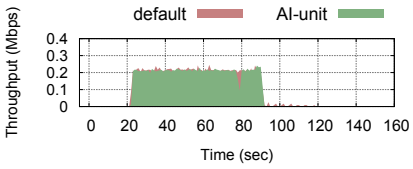


(e) UE 1 allocated slices. Red line indicates a fair sharing scheme with 33% of resources.

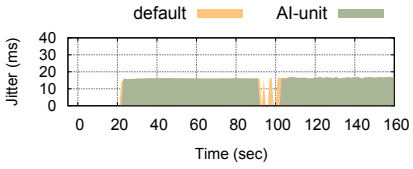
Figure 8: UE 1 QoE with and without the AI unit equipped with CNN-LSTM.



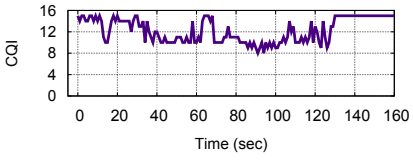
(a) UE 2 interaction with WebRTC service



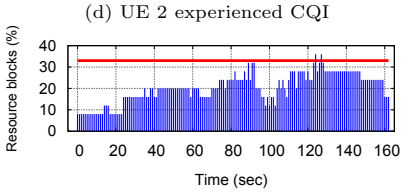
(b) UE 2 interaction with SIPp service



(c) UE 2 experienced Jitter

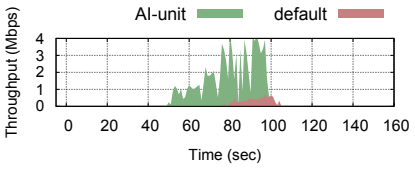


(d) UE 2 experienced CQI

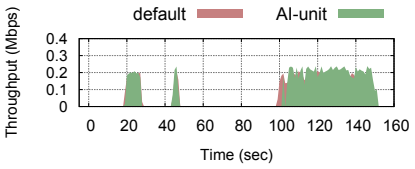


(e) UE 2 allocated slices. Red line indicates a fair sharing scheme with 33% of resources.

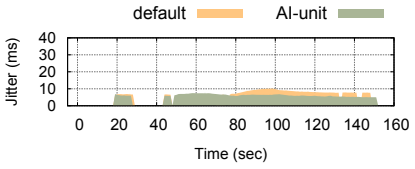
Figure 9: UE 2 QoE with and without the AI unit equipped with CNN-LSTM.



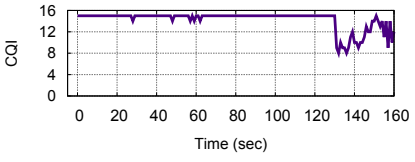
(a) UE 3 interaction with WebRTC service



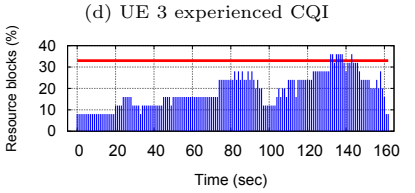
(b) UE 3 interaction with Nginx service



(c) UE 3 experienced Jitter

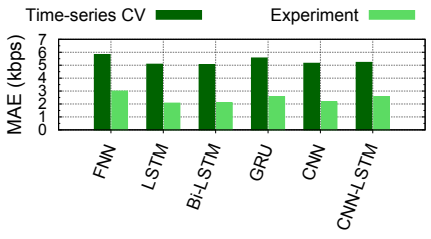


(d) UE 3 experienced CQI

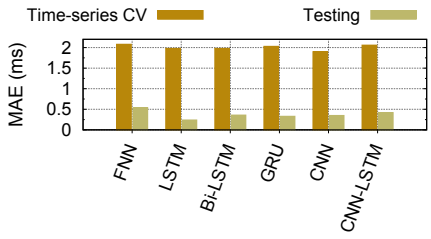


(e) UE 3 allocated slices. Red line indicates a fair sharing scheme with 33% of resources.

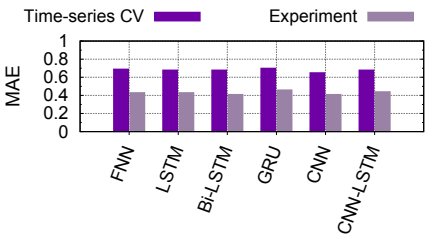
Figure 10: UE 3 QoE with and without the AI unit equipped with CNN-LSTM.



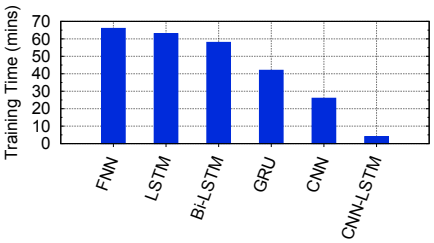
(a) Throughput MAE per Model



(b) Jitter MAE per Model



(c) CQI MAE per Model



(d) Training on Google Colab

Figure 11: Model Off-line Training Evaluation on Google Colab and Experimental Evaluation on Testbed.

Table 4: R^2 Evaluation of the Neural Networks

	FNN	LSTM	Bi-LSTM	GRU	CNN	CNN-LSTM
Time-series CV R^2	0.936	0.940	0.940	0.937	0.945	0.940
Experiment R^2	0.985	0.986	0.987	0.986	0.987	0.986

4.2. Experiment Evaluation

Our real-world experiment on NITOS Testbed evaluates the impact of the AI/ML unit using the CNN-LSTM model on the QoE of the UEs. In Fig. 8-10, we include five different sub-figures for every UE. In specific, the first two subplots (a, b) depict the utilized services, followed by the experienced Jitter (c), then the CQI (d) and subsequently the allocated slices (e) that were provided according to the dynamic slicing allocation algorithm. Noticeably, we compare the resulted QoE of the UEs between the guidance of the AI-unit and the default network configuration. As a "default" configuration, we set all the UE's slices to an equal percentage of 8% during the whole experiment. This is done in an effort to show the results of poor resource management by a fair resource allocation algorithm that provides fixed and equal resources to every UE. Noticeably, a slice of 8% is the minimum that keeps a UE connected to the network in our topology. Moreover, it suffices for the light applications, namely the NGINX and SIPp, on maintaining a high-quality connection. However, the most resource-intensive application, WebRTC, suffers from a lack of resources with a slice of that value. On the other side, a fair algorithm that assigns a slice of 33% to every UE (maximum possible by the default configuration) provides enough resources to all apps but leads to a massive over-provisioning. In specific, it wastes huge amounts of resources for the NGINX and SIPp that could be used to enhance the QoE of the other UEs. Thus, our target is to utilize the AI unit to dynamically and efficiently allocate the slices avoiding over-provisioning to NGINX and SIPp and under-provisioning to the WebRTC.

To begin, the subplots 8a and 8b illustrate the apps used by UE 1. At first, UE 1 interacts with WebRTC until approx. 80 secs, when the Nginx is used in two bursts (70-90 and 120-150 secs). Moreover, the Jitter experienced with the default network slicing is higher initially and gradually decreases, while the CQI fluctuates around low values (6-10) almost during the whole experiment. Noticeably, the algorithm provides the slices on demand by increasing the resource blocks at the maximum of 40% in the first part of the experiment (until 80 seconds), where the demand is clearly higher; the UE interacts with the WebRTC, the Jitter is high and the CQI is poor. Subsequently, the demand declines as the UE 1 switches to the Nginx, the Jitter values decrease and the CQI rises until it plateaus to around 13, at the end of the experiment. Therefore, the slicing percentage gradually decreases until it plunges at the minimum of 8%, after around 140 seconds. This slicing management contributes positively to the QoE of UE 1. Specifically, by comparing the network performance of the default slicing algorithm with the AI-unit's, we can see that the Throughput increased reaching even 1 Mbps with the AI-unit when it used to have around 0.2 Mbps as shown in Fig. 8a. In Fig. 8b, we do not observe any changes since the Nginx is not demanding and it has already reached its peak with the default slicing. Finally, the Jitter falls to lower levels at approx. 10 ms with the guidance of the AI-unit from the 30 ms that it used to be.

Regarding UE 2 (Fig. 9), we see the opposite behavior. In particular, at first UE 2 interacts with the SIPp until approx. 100 seconds, when it switches to the WebRTC. Moreover, the Jitter remains constant during the whole experiment at 15 ms, as shown in Fig. 9c, while the CQI seems to slightly fall at 10-12 values (9d). Noticeably, the algorithms provide a low percentage of resource block at first until around 80 secs, where the demand is relatively low since the quality of the connection is quite good (CQI and Jitter) and the utilized service, the SIPp, is of medium priority. Later, the provided resources are moderately increased to an average of 28% due to the usage of the WebRTC. Importantly, they do not reach higher levels as the link quality is still quite good. Consequently, the QoE of the UE 2 is substantially peaked. Particularly, the WebRTC reaches 4 Mbps Throughput with the AI-unit when it used to reach only a negligible amount of 0.5 Mbps with the default configuration.

Regarding UE 3, the WebRTC is used in the middle part of the experiment, from 50 to approx. 100 secs. Additionally, the Nginx is used majorly in the second part at around 100 secs. The Jitter values are relatively low at an average of 5 ms given that the CQI is extremely high (15) almost during the whole experiment, except for the last 30 secs when it slightly declines to around 11. For these reasons, we observe that the slicing allocation mechanism provides few resources during the first part (no more than 16%) until approx. 70 secs, when the WebRTC Throughput is substantially increased demanding more resources. Then, the algorithm raises the resources to 28% and subsequently drops them to 12% at around 100 secs since the WebRTC is not used anymore. Following that, the slicing scheme gradually increases the resources of the UE 3 until they reach a climax of 36% between approx. 130 to 140 secs in an effort to cope with the drop in the link quality (which is at the lowest level). Generally, the AI-unit assists in the advancement of the QoE since the WebRTC Throughput is increased from 0.5 to 4 Mbps and the Jitter drops from 10 to 5 ms during the second part of the experiment.

Overall, the QoE of all UEs is clearly enhanced given that the Throughput and Jitter performances are ameliorated. Moreover, the slices are provided in a sophisticated way so as to avoid over- and underprovisioning. In fact, this is illustrated in Fig. 8e, 9e, 10e. The red line depicts a value of 33%, which would be the highest slice that could be allocated by a UE with a fixed and fair slicing algorithm. Importantly, our dynamic scheme is able to surpass this limit when the demand for resources is extremely large as well as to decrease the resources dramatically lower than

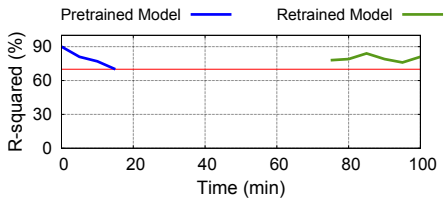


Figure 12: Error before & after online training. The red horizontal line indicates the error threshold.

this percentage when the connection quality is excellent giving, this way, the chance for link improvement to other UEs in the network.

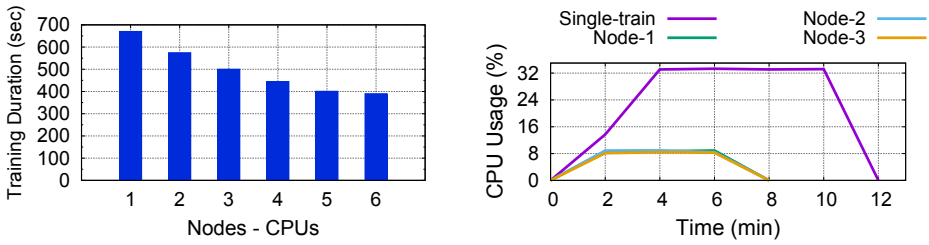
4.3. Online - Distributed Training

To evaluate the MLOps architecture, we run scenarios with new traffic patterns. In specific, we slightly altered the noise distributions in the augmentation steps (sect. 3.2.3) for the new scenarios. In the steps where the standard normal distribution was utilized, we replaced it with an AWGN with a mean of 0 and sd of 1.5. Moreover, we replaced the AWGN with a mean of 0 and an sd of 10 with a new distribution of the same mean but an sd of 15. Thus, we represent a small change in the distribution of the traffic pattern since the baseline patterns still exist in the new scenarios. We noticed that as soon as the new traffic patterns arrived, the predictions deviated quite a bit and the accuracy dropped immediately below the predefined threshold (70%) as shown in Fig. 12. Then, the Kubeflow Pipeline was triggered and started the process of distributed training. In between, the slicing decisions were defined reactively. After the training was over, the updated model started to make predictions again with high accuracy. Noticeably, it converged quite fast with approximately only 20 new samples-scenarios (50 minutes of receiving new samples and updating the model in real-time). It is fast since 300 samples were used for the offline training. Overall, the ability of the scheme to cope with the new patterns relies on many components. First, the differences between the new pattern distribution with the one that the model has converged previously. The bigger the difference the larger the number of new samples required. Further, the processing power of the infrastructure is vital. For instance, Graphics Processing Units (GPUs) and TPUs outperform CPUs substantially accelerating the updating.

To evaluate our distributed training architecture we scale our cluster up to six NITOS nodes that carry octa-core processors (Intel-Core i7-3770 at 3.40 GHz Processor). Observing Fig. 13a, the increase in performance is almost linear as the training time seems to converge at 6 CPUs succeeding in reducing training time by half. This optimization of training time enables us to train the model as quickly as possible and to be able to cope more accurately with the predictions of the most recent data of traffic and network conditions. It is worth noting that the training data were taken from a sample of the entire dataset: 20 scenarios with 18 columns-features. The distributed training is applied to our cluster (NITOS Testbed) where only CPUs are used and the purpose of this experiment was to show how beneficial it is to use all resources simultaneously in the case of online training. The CNN-LSTM model was employed for the experiment. Performance can be further enhanced by utilizing a GPU cluster. In addition, load balancing is ensured in our cluster as illustrated in 13b. In this experiment, we compared the CPU usage for the training of the model between a single machine-container and distributed 3 pods - 3 nodes synchronous all-reduce training. We notice that the single pod has almost 4 times CPU usage compared to the distributed pods which consume resources evenly in the cluster. These measurements were taken from the Prometheus adapter which we integrated into the cluster for resource monitoring.

5. Limitations and Discussions

While our results add valuable insights to the evolving domain of slicing in cloud-native 5G Networks, it's important to recognize the limitations of our infrastructure. The constraint on the number of UEs, capped at three, was a practical consideration due to the challenges associated with establishing connections in our real telecommunications network setup. The setup operates as a private 5G network where the application usage is more static, meaning the variety of applications that the users interact with, is relatively fixed. This may not fully represent the dynamic nature of application usage in public 5G networks, where applications with different network requirements may be in use simultaneously. To address this, extensive datasets that capture a wide range of user behaviors, application interactions, and network patterns are essential. These datasets will serve as the foundation for training machine learning models and refining the slice allocation algorithm to handle the intricacies of dynamic application usage in public 5G networks. Also, by increasing the scale of the experimental setup by connecting a larger number of end devices is crucial to emulate the complexities of public networks. This expansion allows for a more comprehensive evaluation of the slice allocation mechanism's performance in diverse and dynamic scenarios. Nevertheless, the service-aware slice allocation mechanism provides an end-to-end solution that can be directly plugged into any type of telecommunication network, regardless of the operator. From a performance perspective, there can be limitations concerning the real-time packet inspection and classification, as the overall cell throughput is increased. Such limitations can be easily overcome, when employing data-plane traffic accelerators in the network, for bypassing the operating system stack and providing direct access to the network. Implementations of libraries such as DPDK, enhanced Berkeley Packet Filters (eBPF) or employing a Vector Packet Processing (VPP) methodology



(a) Dist. Training Duration

(b) Dist. CPU Usage

Figure 13: Experimental results for Distributed Training

in the packet handling can offer significant gains in performance, especially in the cases where the overall network traffic reaching the UPF surpasses 1Gbps. The aforementioned limitations provide avenues for future work, including extending the experimentation to larger-scale setups, exploring the performance of the slice allocation mechanism in public 5G networks with dynamic application usage, and investigating solutions to handle multiple UEs.

6. Conclusion

In this work, we developed and experimentally evaluated an ML-driven approach for defining the optimal slice application in the cellular 5G network, based on the applications that are hosted on top. Our framework can autonomously decide on the allocations, based on the ML-driven classification of the traffic and the mobility of users, providing near-real-time performance. The selection of the ML model was determined after experimenting with several neural network-based approaches, with the one performing optimally being a CNN-LSTM stacked model for our data. The solution is able to analyze and classify traffic from different applications correctly. At the same time, it considers the user's connection quality, and appropriately enforces the slices in the network. In the future, we foresee wrapping parts of our contribution into xApps and porting our solution to the O-RAN architecture. The detailed implementation instructions and code repository can be accessed on GitHub: GitHub.[1]. Additionally, partial datasets and code configurations for the framework are provided in [42].

Acknowledgement

The research leading to these results has received funding from the European Union's Horizon Europe Research and Innovation Programme for research, technological development, and demonstration under Grant Agreement Number No 101079774 (Horizon Europe SLICES-PP) and the European Union's Horizon 2020 research and innovation programme under grant agreement No 101008468 (SLICES-SC). The European Union and its agencies are not liable or otherwise responsible for the contents of this document; its content reflects the view of its authors only.

[1]For specifics on the experimental setup, refer to: `https://github.com/teo-tsou/app_aware_5g`

References

[1] N. Kato, B. Mao, F. Tang, Y. Kawamoto, and J. Liu, "Ten Challenges in Advancing Machine Learning Technologies toward 6G," *IEEE Wireless Communications*, vol. 27, no. 3, pp. 96–103, 2020.

[2] I. Tomkos, D. Klonidis, E. Pikasis, and S. Theodoridis, "Toward the 6G Network Era: Opportunities and Challenges," *IT Professional*, vol. 22, no. 1, pp. 34–38, 2020.

[3] C.-Y. Chang, N. Nikaein, O. Arouk, K. Katsalis, A. Ksentini, T. Turletti, and K. Samdanis, "Slice Orchestration for Multi-Service Disaggregated Ultra-Dense RANs," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 70–77, 2018.

[4] C. Ge, N. Wang, S. Skillman, G. Foster, and Y. Cao, "QoE-Driven DASH Video Caching and Adaptation at 5G Mobile Edge," in *ACM Conference on Information-Centric Networking*, ser. ACM-ICN '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 237–242. [Online]. Available: https://doi.org/10.1145/2984356.2988522

[5] R. Smith, C. Freeberg, T. Machacek, and V. Ramaswamy, "An O-RAN Approach to Spectrum Sharing Between Commercial 5G and Government Satellite Systems," in *IEEE Military Communications Conference (MILCOM)*, 2021, pp. 739–744.

[6] L. M. Larsen, A. Checko, and H. L. Christiansen, "A survey of the functional splits proposed for 5G mobile crosshaul networks," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 146–172, 2018.

[7] A. Garcia-Saavedra and X. Costa-Perez, "O-RAN: Disrupting the virtualized RAN ecosystem," *IEEE Communications Standards Magazine*, 2021.

[8] A. Ghosh, A. MÃ¤der, M. Baker, and D. Chandramouli, "5G Evolution: A View on 5G Cellular Technology Beyond 3GPP Release 15," *IEEE Access*, vol. PP, pp. 1–1, 09 2019.

[9] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vNF placement at the network edge," in *IEEE conference on computer communications (INFOCOM)*. IEEE, 2018, pp. 693–701.

[10] Z. Xu, X. Zhang, S. Yu, and J. Zhang, "Energy-Efficient Virtual Network Function Placement in Telecom Networks," in *International Conference on Communications (ICC)*, 2018, pp. 1–7.

[11] I. Sarrigiannis, K. Ramantas, E. Kartsakli, P.-V. Mekikis, A. Antonopoulos, and C. Verikoukis, "Online VNF Lifecycle Management in an MEC-Enabled 5G IoT Architecture," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4183–4194, 2020.

[12] X. Fei, F. Liu, H. Xu, and H. Jin, "Towards load-balanced VNF assignment in geo-distributed NFV Infrastructure," in *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2017, pp. 1–10.

[13] ——, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2018, pp. 486–494.

[14] D. B. Oljira, K.-J. Grinnemo, J. Taheri, and A. Brunstrom, "A model for QoS-aware VNF placement and provisioning," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017, pp. 1–7.

[15] L. Qu, C. Assi, and K. Shaban, "Delay-Aware Scheduling and Resource Optimization With Network Function Virtualization," *IEEE Transactions on Communications*, vol. 64, no. 9, pp. 3746–3758, 2016.

[16] D. Kumar, S. Chakrabarti, A. S. Rajan, and J. Huang, "Scaling Telecom Core Network Functions in Public Cloud Infrastructure," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2020, pp. 9–16.

[17] I. Alawe, A. Ksentini, Y. Hadjadj-Aoul, and P. Bertin, "Improving Traffic Forecasting for 5G Core Network Scalability: A Machine Learning Approach," *IEEE Network*, vol. 32, no. 6, pp. 42–49, 2018.

[18] I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, P. Bertin, and D. Darche, "On the scalability of 5G core network: The AMF case," in *IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2018, pp. 1–6.

[19] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks," in *International on Conference on emerging Networking EXperiments and Technologies (CONEXT)*, 2016, pp. 427–441.

[20] R. Schmidt, M. Irazabal, and N. Nikaein, "Flexric: An sdk for next-generation sd-rans," in *CONEXT 2021, 17th International Conference on Emerging Networking EXperiments and Technologies, 7-10 December 2021, Munich, Germany (Virtual Conference)*, ACM, Ed., Munich, 2021.

[21] ITRI, "Athena Orchestrator - O-RAN SMO RIC, note=[Online], https://event.itri.org/CES2023/tech_details/22."

[22] C.-C. Chen, C.-Y. Chang, and N. Nikaein, "Flexslice: Flexible and real-time programmable ran slicing framework," in *GLOBECOM 2023, IEEE Global Communications Conference, 4-8 December 2023, Kuala Lumpur, Malaysia*, Kuala Lumpur, 2023.

[23] J. Kaur, M. A. Khan, M. Iftikhar, M. Imran, and Q. E. U. Haq, "Machine Learning techniques for 5G and beyond," *IEEE Access*, vol. 9, pp. 23 472–23 488, 2021.

[24] O. Aouedi, K. Piamrat, S. Hamma, and J. Perera, "Network traffic analysis using machine learning: an unsupervised approach to understand and slice your network," *annals of telecommunications*, 11 2021.

[25] Q. Qin, K. Poularakis, K. K. Leung, and L. Tassiulas, "Line-speed and scalable intrusion detection at the network edge via federated learning," in *IFIP Networking Conference (Networking)*, 2020, pp. 352–360.

[26] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 104–117, 2013.

[27] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, ser. MineNet '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 281–286. [Online]. Available: https://doi.org/10.1145/1162678.1162679

[28] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2014.

[29] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[30] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 76–81, 2019.

[31] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE Network*, vol. 26, no. 1, pp. 35–40, 2012.

[32] C. Jiang, H. Zhang, Y. Ren, Z. Han, K.-C. Chen, and L. Hanzo, "Machine Learning paradigms for Next-Generation Wireless Networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 98–105, 2016.

[33] M. Dryjański, Kułacz, and A. Kliks, "Toward Modular and Flexible Open RAN Implementations in 6G Networks: Traffic Steering Use Case and O-RAN xApps," *Sensors*, vol. 21, no. 24, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/24/8173

[34] A. Thantharate, A. V. Tondwalkar, C. Beard, and A. Kwasinski, "Eco6g: Energy and cost analysis for network slicing deployment in beyond 5g networks," *Sensors*, vol. 22, no. 22, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/22/8614

[35] B. Brik and A. Ksentini, "On Predicting Service-oriented Network Slices Performances in 5G: A Federated Learning Approach," in *IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2020, pp. 164–171.

[36] C.-Y. Chang and N. Nikaein, "Closing in on 5G control apps: enabling multiservice programmability in a disaggregated radio access network," *IEEE Vehicular Technology Magazine*, vol. 13, no. 4, pp. 80–93, 2018.

[37] Q. Liu, N. Choi, and T. Han, "OnSlicing: Online End-to-End Network Slicing with Reinforcement Learning," in *International Conference on Emerging Networking EXperiments and Technologies (CONEXT*, ser. CoNEXT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 141–153. [Online]. Available: https://doi.org/10.1145/3485983.3494850

[38] S. Ravindran, S. Chaudhuri, J. Bapat, and D. Das, "Novel adaptive multi-user multi-services scheduling to enhance throughput in 5g-advanced and beyond," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2024.

[39] N. Salhab, R. Langar, and R. Rahim, "5G network slices resource orchestration using Machine Learning techniques," *Computer Networks*, vol. 188, p. 107829, 2021.

[40] N. Makris, C. Zarafetas, S. Kechagias, T. Korakis, I. Seskar, and L. Tassiulas, "Enabling open access to LTE network components; the NITOS testbed paradigm," in *IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–6.

[41] I. Chatzistefanidis, N. Makris, V. Passas, and T. Korakis, "UE Statistics Time-Series (CQI) in LTE Networks," 2022. [Online]. Available: https://dx.doi.org/10.21227/ec7p-xq38

[42] T. Tsourdinis, I. Chatzistefanidis, N. Makris, and T. Korakis, "Ue network traffic time-series (applications, throughput, latency, cqi) in lte/5g networks," 2022. [Online]. Available: https://dx.doi.org/10.21227/4ars-fs38

[43] D. Golubovic and R. Rocha, "Training and Serving ML workloads with Kubeflow at CERN," in *EPJ Web of Conferences*, vol. 251. EDP Sciences, 2021, p. 02067.

[44] D. Green, "Pyshark: Python wrapper for tshark, allowing python packet parsing using wireshark dissectors." [Online], https://github.com/KimiNewt/pyshark.

[45] C. Chen, W. Wang, and B. Li, "Round-Robin Synchronization: Mitigating Communication Bottlenecks in Parameter Servers," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 532–540.