ELASTIC ORCHESTRATION FRAMEWORK FOR SON-ENABLED 5G NETWORKS USING MACHINE LEARNING

KHALID ALI

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTERS OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGIES
YORK UNIVERSITY
TORONTO, ONTARIO
AUGUST 2022

Abstract

5G networks are expected to support a variety of services and applications by having a more stringent latency, reliability, and bandwidth requirements compared to previous generations. To meet these requirements, Open Radio Access Networks (O-RAN) has been proposed. The *O-RAN Alliance* assumes O-RAN components to be Virtualized Network Functions (VNFs). Furthermore, O-RAN allows employing Machine Learning (ML) solutions to tackle challenges in resource management. However, intelligently managing resources for O-RAN can prove challenging. Network providers need to dynamically scale resources in response to incoming traffic. Elastically allocating resources provides higher flexibility, reduces OPerational EXpenditure (OPEX), and increases resource utilization. In this work, we propose and evaluate an elastic VNF orchestration framework for O-RAN. The proposed system consists of a traffic forecasting-based dynamic scaling scheme using ML, and a Reinforcement Learning (RL) based VNF placement policy. The models are evaluated based on their predictive capabilities subject to all Service-Level Agreements.

Acknowledgements

First and foremost, I would like to extend my deepest and sincerest gratitude to my supervisor Dr. Manar Jammal, without whom this thesis would not have been possible. Dr. Manar's guidance and constant concern, not only for my academic progress but also for my general well-being, have played a pivotal role in making this work a reality. Her editorial rigor along with her patience and constructive feedback have allowed me the time and space to develop the necessary technical background required to make this thesis possible.

I would like to furthermore extend my thanks to my friends, Saffia and Mohamed, for their constant and unwavering support, always pushing me to put in the time and effort required. My work is substantially stronger thanks to their intellectual and emotional support. I would also like to state my appreciation for my friends, Tarek and Ibrahim, for encouraging me to pursue master's at York University.

Finally, I would like to dedicate this work to Nadia Elsadik, who instilled in me a love of knowledge, of research, and of the pursuit of truth. Words of thanks and gratitude do not come close in matching the role she played in bringing this thesis to fruition. This process has involved many difficult, demanding, and also wonderfully stimulating days. The bud may have a bitter taste, but sweet will be the flower.

Table of Contents

List of Tables

List of Figures

Preface

The research presented in this thesis is the original work of Khalid Ali, and it has been conducted in collaboration with Ciena Canada and the Ontario Centre of Excellence led by Dr. Manar Jammal. This thesis is organized in paper format following the guidelines for paper-based theses. Parts of this report have been accepted to the peer-reviewed publication listed below.

- Ali, K. & Jammal, M. (2022). Traffic Forecasting for Open Radio Access Networks Virtualized Network Functions in 5G Networks. World Academy of Science, Engineering and Technology, Open Science Index 186, International Conference on Computing Communication and Networking Technologies, 16(8).

Parts of this report are still undergoing review by the following venues:

- Ali, K. & Jammal, M. (2022). Proactive VNF Scaling and Placement in 5G O-RAN using ML. (submitted)

- Ali, K. & Jammal, M. (2022). Machine Learning in 5G Open Radio Access Networks: Challenges and Opportunities. (submitted)

Abbreviations

5G	5th Generation
A2C	Advantage Actor Critic
ACF	Autocorrelation Function
ADF	Augmented Dickey Fuller
ARIMA	Auto Regressive Integrated Moving Average
BS	Base Station
CAPEX	Capital Expenditure
CNN	Convolutional Neural Network
COTS	Commercial-Of-The-Shelf
CP	Cloud Provider
CPU	Central Processing Unit
CRAN	Cloud Radio Access Network
eMBB	enhanced Mobile Broadband
GNN	Graph Neural Network

HDD	Hard Disk Drive
HO	HandOver
ILP	Integer Linear Programming
IoT	Internet of Things
LSTM	Long-Short Term Memory
MEC	Mobile Edge Computing
MIMO	Multiple-Input Multiple-Output
ML	Machine Learning
MLP	Multi Level Perceptron
mMTC	massive Machine Type Communication
NFV	Network Functions Virtualization
O-CU	Open Centralized Unit
O-DU	Open Distributed Unit
O-RAN	Open Radio Access Network
O-RU	Open Radio Unit
OPEX	Operational Expenditure
PACF	Partial Autocorrelation Function
PPO	Proximal Policy Optimization
PU	Processing Unit
QoE	Quality of Experience

QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
RF	Radio Frequency
RIC	RAN Intelligence Controller
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SDN	Software-Defined Networking
SFC	Service Function Chain
SLA	Service-Level Agreement
SMO	Service Management and Orchestration
SON	Self-Organizing Networks
TRPO	Trust Region Policy Optimization
UE	User Equipment
VM	Virtual Machine
VNF	Virtual Network Function
VNF-OP	Virtual Network Function Orchestration Problem
WAN	Wide Area Network
uRLLC	ultra-Reliable Low Latency Communications
xAPP	μservice Application

1 Introduction

In the past decade, the surge in demand for advanced real-time applications and massive Internet of Things (IoT) structure has resulted in a significant burden on the wireless communication infrastructure. Furthermore, the 5th Generation of mobile communications (5G) enabled massive IoT devices connectivity that requires large coverage with low throughput and low power consumption. Moreover, Mobile Edge Computing (MEC) reduces the congestion over the mobile network by bringing the processes closer to the end user. MEC also contributes to reducing the delay in the 5G networks. The lower latency allows for applications with higher speed requirements. Typically, there are 3 families of 5G use cases: enhanced Mobile Broadband (eMBB), massive Machine-Type Communication (mMTC), and ultra-Reliable Low-Latency Communication (uRLLC). 5G is expected to support a variety of services and applications by having a more stringent latency requirement and consuming a significantly higher bandwidth compared to the previous generations. These applications include but not limited to massive IoT, MEC, vir-

tual reality, and industry specific applications in healthcare, retail, agriculture, and manufacturing. However, traditional Radio Access Networks (RANs) are deemed to be inflexible with regard to the 5G due to its stringent requirements. Moreover, the cost associated with operating real-time services and applications has risen drastically. To overcome these challenges, the *O-RAN Alliance* has introduced an Open RAN (O-RAN) solution, which allows mobile network operators to define their services using components from multiple vendors. O-RAN tackles the challenges mentioned through incorporating cloud, virtualization, and intelligence into the existing architecture. Additionally, thanks to openness and virtualization of O-RAN combined with its latency-aware architecture, it reduces the deployment and maintenance costs by exploiting state-of-the-art intelligent solutions to realize a Self-Organizing Network (SON) [1]. For instance, telecom operators provision a vast amount of resources, expending significant operational and maintenance costs, in addition to its inflexibility in terms of scalability [2]. Hence, they are deemed impractical and inefficient regarding the 5G applications. Many Machine Learning (ML) solutions have been proposed to enhance O-RAN functionalities, such as Q-learning for resource management and allocation, and Bayesian learning in Multiple-Input Multiple-Output (MIMO) network channel application.

Network Functions Virtualization (NFV) has removed the limitations on network evolution through decoupling the network functions from the network ap-

pliances and deploying them on Commercial Off-The-Shelf (COTS) hardware [3]. The decoupling results in a reduction in the OPerational and CApital EXpenditure (OPEX/CAPEX) [4]; providers have registered 49% CAPEX savings compared to traditional deployment [5]. Additionally, it enables exploiting state-of-the-art ML solutions that reduce OPEX. OPEX is generally modeled based on the costs associated with operating the Virtual Network Functions (VNFs); it consists of the cost of running a VNF, deploying a VNF, backup VNFs, and forwarding the traffic. To reduce these costs, resource management and allocation techniques are employed. When it comes to 5G networks, the *O-RAN Alliance* assumes the O-RAN components, such as the Distributed Unit (DU), Control Unit (CU), and the Near-Real Time RAN Intelligent Controller (Near-RT RIC), to be virtualized and can be treated as VNFs [6]. The O-RAN VNFs then form the O-RAN Service Function Chain (SFC) shown in Figure 1.1. Hence, NFV technology can be exploited to reduce the cost of running those VNFs, making it more sustainable and agile while maintaining Service-Level Agreements (SLAs).

When it comes to resource allocation, the research problems that arise are server placement, function placement, and dynamic resource management. The server and function placement problems are explored thoroughly in the literature. In contrast, the dynamic resource management problem in NFV is less saturated. Dynamic resource management can be defined as dynamically scaling the resources allocated

for VNFs in accordance with real-time network demand [7]. In a nutshell, the more resources allocated for a VNF in an SFC, the more traffic that VNF can handle. However, over-provisioning resources can result in a low utilization level due to traffic dynamicity, thus, increasing the cost of those VNFs. The more efficient approach to the resource management problem is to dynamically allocate the resources in response to the traffic demand. Such systems, capable of dynamically allocating resources according to the traffic, are called elastic.

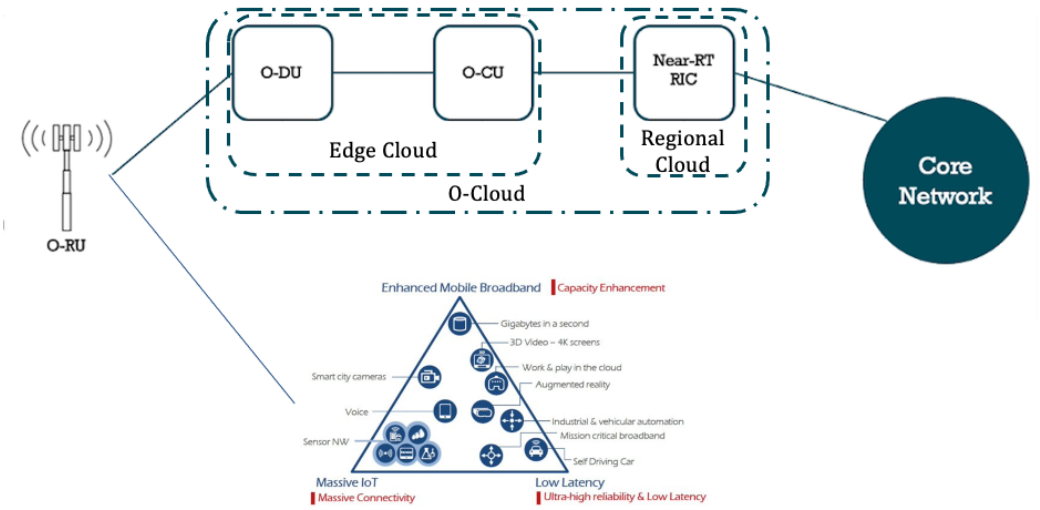

Figure 1.1: Simple O-RAN service function chain (SFC).

1.1 Elasticity

Elasticity can be defined as the degree to which a system is capable of adapting to a varying workload by provisioning and deprovisioning resources autonomously, such that at each point in time the available resources match the current demand as closely as possible [8]. Elasticity is usually associated with the concept of scalability of the system; however, both are not equivalent. Elasticity is a tactical resource allocation operation. It provides the necessary resources required for the current task and handles varying loads for short periods. For elasticity, time is a central aspect; it depends on the speed of response to a certain changed workload. In contrast, scalability is a strategic resource allocation operation. Scalability handles the increase and decrease of resources according to the system's workload demands. Scalability is a time-free notion, and it does not capture how long it takes for the system to attain the required performance level. Elastic models are evaluated based on key metrics such as capacity, Quality of Service (QoS), and resource utilization. Generally, these metrics measure idleness and over- and under-utilization.

On a practical sense, elasticity can be achieved through two main methods: horizontally scaling by adding or removing instances, or vertically scaling by adding or removing resources from an instance. However, triggering one of these methods tends to be quite challenging. Elastic systems can trigger scaling either reactively

or proactively (Figure 1.2). Reactive strategies are relatively simpler to implement and provide a good performance, however, since they react to already arrived traffic, they result in a significant delay. This delay stems from copying the VNF instance and instantiating new VNF instances whenever a scaling action is triggered, thus disrupting existing connections, which can result in an SLA violation [7]. On the other hand, proactive techniques help in reducing the cost while maintaining the QoS. Since they predict the demand beforehand and proactively scale resources, they can render the delay to be irrelevant. However, they can be complex to implement.

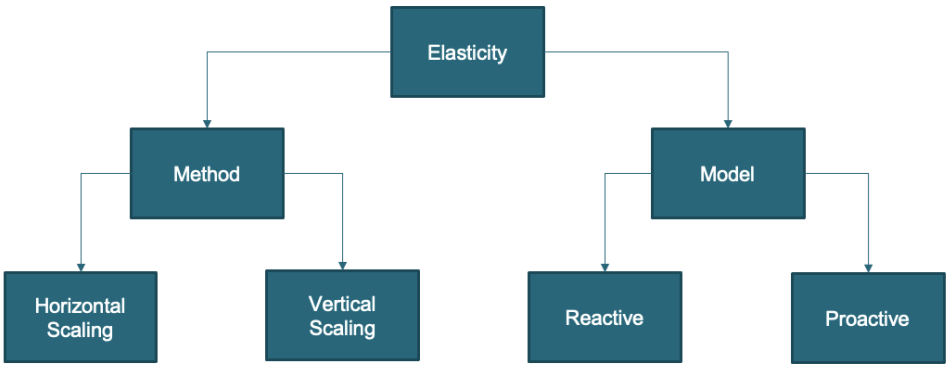

Figure 1.2: Elasticity methods and models.

1.2 Motivation and Objectives

The 5G evolution and the O-RAN architecture have paved the way for realizing RANs as an SON. SON can be defined as an adaptive and autonomous network that is also scalable, stable, and agile enough to maintain its desired objectives. The main objectives of SON are to provide intelligence inside the network. The added intelligence aims to facilitate the work of operators and provide network resilience. Furthermore, it aims to reduce the overall complexity, CAPEX and OPEX, and to simplify the coordination, optimization, and configuration procedures of the network. SON can be divided into three categories: self-configuration, self-optimization, and self-healing. The *O-RAN Alliance* assuming that O-RAN components are VNF has paved the way for exploiting intelligent frameworks to realize O-RAN as a self-healing network. One of the challenging aspects of realizing SONs is autonomous resource management. As mentioned, system capable of autonomously allocating resources are called elastic. Therefore, the problem we are exploring in this work is developing an intelligent elastic solution for O-RAN VNFs to contribute in realizing it as an SON.

Several elastic models are proposed in the literature aiming to elastically scale resources. However, only a few considered a proactive approach, as opposed to a reactive one. Nonetheless, many recent efforts focused on employing an ML-based

proactive approach. Since O-RAN puts a significant emphasis on latency in the network, proactive elastic models are more suitable. Simply put, if one can successfully predict the traffic, new VNF deployment can be done in advance, thus rendering the deployment delay associated with reactive models irrelevant, in addition to maintaining SLAs [2]. Elasticity-aware ML models are a prime candidate in addressing the dynamicity and complexity present within the cloud environment. Hence, the aim of this work is to address this gap by designing and implementing a proactive Elastic O-RAN VNFs orchestration policy for 5G networks. To achieve a proactive elastic policy, there are 3 main components the system must have; a proactive element that predicts future behavior, a scaling scheme that maps the prediction onto an executable action, and provisioning element that can execute the action. Moreover, all these components have to work in an autonomous manner. With that in mind, we formulate the main objectives of the thesis as follows:

- Introduce dynamic network conditions to the O-RAN units placement problem, i.e., the dynamicity of the traffic.
- Contribute to data preparation for O-RAN through compiling a training dataset based on realistic network conditions to be used for ML models implementation.
- Introduce a scaling policy that can leverage ML to allocate resources based

on the incoming demand.

- Introduce a Reinforcement Learning agent that can handle dynamically deploying instances based on the scaling policy while maintaining the SLAs.
- Replace the current existing reactive models by introducing a proactive model, i.e., traffic forecasting.
- Implement a proactive Elastic orchestration system for dynamic resource allocation in a 5G O-RAN environment that reduces the OPEX by increasing utilization and reducing resource fragmentation.

1.3 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 provides some background; presenting a brief overview of O-RAN and its architecture, and discussing the related work published in the area of VNF orchestration and resource management, highlighting limitations of existing work. Chapter 3 presents the problem formulation and provides a high-level discussion of the proposed framework. Chapter 4 explains the proposed methodology and its block diagram, discussing the requirements for each stage. Chapter 5 elaborates on the models' implementation in each stage and their performance evaluation. Finally, Chapter 6 lists our contributions and elaborates on the possible future directions for this research.

2 Background

In this Chapter, we first present a brief overview of O-RAN, explaining its components, architecture, and O-RAN as an SON, in addition to the various benefits O-RAN architecture provides. Moreover, we explore applicability of Intelligent solutions in O-RAN architecture. Afterwards, an overview of existing work in the literature is presented, focusing on the main contributions and limitations of current solutions portrait in the literature.

2.1 O-RAN Overview

RANs are a major component of wireless communication systems. They serve as an intermediary platform that connects the User Equipment (UE) with core network. RANs typically consist of two main units: a Radio Unit (RU) consisting of a transceiver antenna responsible for transmitting and receiving Radio Frequency (RF) signals, and a Processing Unit (PU) responsible for resource utilization, radio management, encryption, precoding, etc. [9].

Traditionally, RANs have integrated the RU and the PU in a Base Station (BS). Initially, the number of users and the data rates are much less compared to current numbers. Moreover, thanks to employing frequency reuse frameworks, there was little to no need for computation resources employed in filtering interferences. Recently, as data-hungry applications, such as Virtual Reality, are introduced and the amount of UEs increases; the demand for densification has surged. However, the densification alone is not enough to fulfill the demand for huge data rates. To address this issue, researchers started employing mmWave communication, which in turn have raised the demand for a connected framework. Thus, resulting in the introduction of what is called Cloud RAN (CRAN). In CRAN, the PUs in all the BSs are aggregated together into a centralized Cloud Processor (CP) [9].

The evolution of RANs is not only on the architectural level, but also on the level of specific functionalities. For instance, an example of such evolution is the shift from BS centric approach to a UE centric approach in the process of selecting which BS serves a specific UE. Another example is the addition of extra antennas to the RU. This is done to mitigate the attenuation and the poor diffraction of the mmWaves. Through introducing beams in the direction of the user (Beamforming), a more reliable mmWave communication is achieved.

The recent RANs are divided into an RU in each BS and a centralized CP. However, due to the introduction of applications that require ultra-low latency and

high reliability; the internal architecture of the RANs has been disaggregated based on functionality [10]. The disaggregated units include:

- Radio Function: it is located at the RU and responsible for performing physical layer operations such as digital to analog conversion, modulation, amplification, etc.
- Baseband Processing Function: it is responsible for upper layer functions such as carrier aggregation, radio scheduling, beam formation, and antenna selection.
- Radio Control Function: the unit oversees the resource allocation and distribution as well as load sharing. This unit is one of the most essential units in RANs as it is responsible for performing virtualization and resource management.
- Packet Switching Function: this unit, like the Radio Control Function, performs key functionalities related to virtualization. Specifically, it oversees the multi-path handling, scheduling, and encryption.

The ultimate goal of the RAN is to connect the UEs through the RUs to the core network. The traditional RAN does this with no regard for the applications or services. In contrast, the new generation of RAN, O-RAN, takes the applications into account. This results in added agility to the network. This results in an added

agility to the network, however, more layers are added, thus increasing latency and computational power requirements; thus, resulting in a highly complex RAN. To overcome these challenges, modern ML techniques in addition to Mobile Edge Computing (MEC) capabilities are employed in O-RAN infrastructure.

The distribution of functionality in the O-RAN increases the reliability by avoiding a single node failure. Additionally, in O-RAN, the separation of the Control Plane and the User Plane allows the implementation on a server platform. Furthermore, the fact that they are intended to work separately enables scalability and increases flexibility in the O-RAN. Simply put, the O-RAN is a virtualized, application specific, and software-oriented RAN; thus, enabling it to support high-speed applications and networks [10].

2.1.1 O-RAN Architecture

The O-RAN architecture (Figure 2.1 [10]) is based on decoupling the non-real-time functionalities and the real-time ones. Through this decoupling, the RAN Intelligent Controller (RIC) is introduced. RIC contributes to lowering the infrastructural and operational cost for network operators. Furthermore, they improve the performance and the agility of the RAN. They provide advanced ML enabled functionalities to increase efficiency and improve radio resource management. Furthermore, RICs host ML functionalities; the service and the process of training models

are hosted in the non-Real Time RIC (non-RT RIC), and the trained models are hosted in the near-Real Time RIC (near-RT RIC). The near-RT RIC is a near-RT μservice based platforms for hosting μservice applications (xAPPs). These xAPPs range from controlling a group of RAN infrastructure (CU, DU, eNB) to management and self-optimization of the RAN, many of which are ML-based. Through utilizing the available RAN data, the xAPPs can improve the Quality of Experience (QoE) and increase resource utilization.

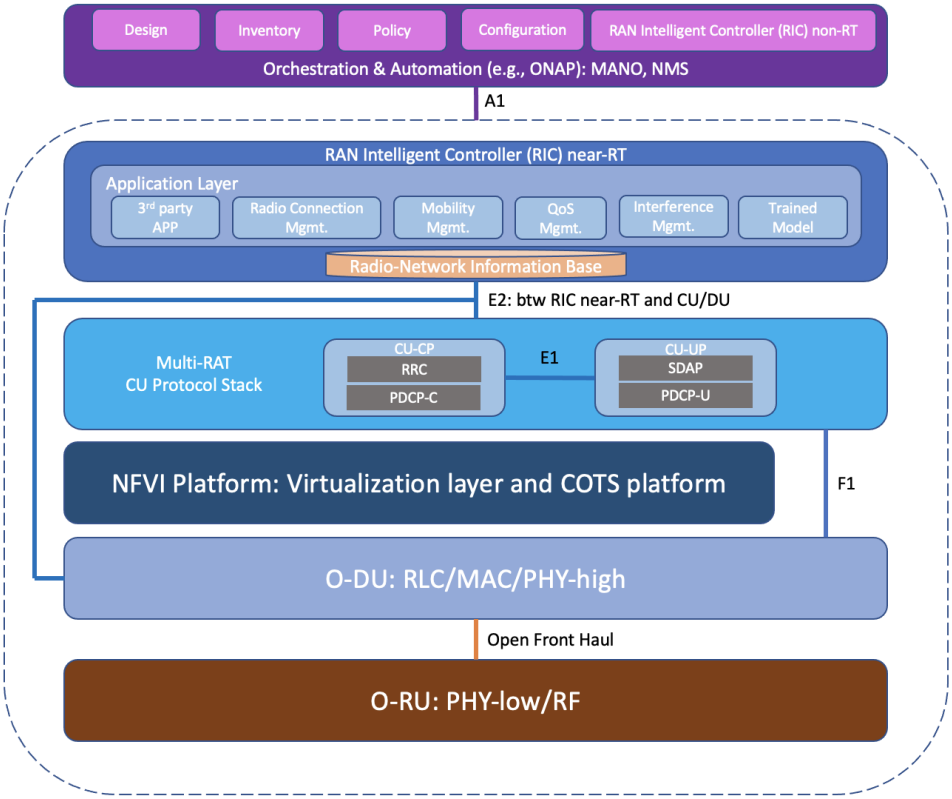

Figure 2.1: O-RAN architecture.

The near-RT RIC contains the databases related to the radio network information base, which tracks the performance of the network through the E2 and A1 interfaces. The E2 interface is mainly concerned with relaying Radio Resource Management related metrics from the O-CU and the O-DU to the near-RT RIC. In turns, the near-RT RIC provides a kind of Radio Management and Orchestration through the ML models it hosts. Additionally, the near-RT RIC is also responsible for the HandOver (HO) operation and QoS management. The A1 interface is responsible for relaying the ML-based policies and training models to the non-RT RIC. In a nutshell, the non-RT RIC is responsible for supporting the non-RT intelligence control and resource management and providing guidance and support to the near-RT RIC [10].

The O-CU is responsible for supporting higher RAN functions, such as Radio Resource Control (RRC) and Packet Data Convergence Protocol (PDCP) protocols. The near-RT RIC controls basic functions such as HO and virtualizations which provides the ability to distribute over elements. O-DU and O-RU oversee radio functions and processing, baseband processing, etc. In particular, the O-RU is in charge of hosting lower physical (LoPhy) functions, while O-DU is responsible for executing higher physical (HiPhy) functions, Media Access Control (MAC) procedures and radio control tasks. O-RUs are located as close as possible to antennas, while O-DUs are generally higher up in the network [11].

The decoupling of software and hardware in O-RAN results in a unified architecture. Such architecture proves beneficial in terms of low latency and allows for network slicing among others. Some benefits of O-RAN architecture are:

- Agility: the introduction of the NFV enables the network to be suitable for future and existing generations of RAN.
- Deployment Flexibility: the Software-Defined Networking (SDN) employed in O-RAN makes it flexible in terms of upgrade and installation.
- Real-Time Responsive: since the O-RAN is application-driven, the network behaves on the basis of the service requested. This makes it require a very low latency over the less critical services.
- Operating Cost Reduction: since O-RAN utilizes COTS hardware and employs modern ML techniques, the cost reduction estimation goes as high as 80% in maintenance and operating costs.

2.1.2 Intelligence in O-RAN

The 5G evolution and the O-RAN architecture have paved the way for realizing the RAN as an SON. SON can be defined as an adaptive and autonomous network that is also scalable, stable, and agile enough to maintain its desired objectives. The main objectives of SON are to provide intelligence inside the network. The added

intelligence aims to facilitate the work of operators and provide network resilience. Furthermore, it aims to reduce the overall complexity, CAPEX and OPEX, and to simplify the coordination, optimization, and configuration procedures of the network. SON can be divided into three categories:

- Self-configuration: all the configuration procedures necessary to make the network operable (e.g., configuring operational parameters, radio parameters, and NCL).
- Self-optimization: the functions which continuously optimize the BSs and network parameters to guarantee a near-optimal performance (e.g., load balancing, resource optimization, caching, etc.).
- Self-healing: continuously monitors the system to ensure a fast and seamless recovery by detecting the failure events, diagnosing the failure, and triggering the appropriate compensation mechanisms (e.g., fault detection, fault classification, and outage management).

To realize the O-RAN as SON, many of the state-of-the-art technologies have been employed in O-RAN architecture such as IoT services, ML, and MEC.

- IoT enabled network: O-RAN deployment has removed the connectivity restrictions around the IoT applications, hence, resulting in the possibility of implementing some new applications in the e-health and security fields. In these

applications, the IoT devices are to be connected over the O-RAN framework. Additionally, the disaggregation of the non-RT RIC has enabled massive IoT devices connectivity that requires large coverage with low throughput and low power consumption.

- MEC enabled network: MEC is defined as an architecture that allows edge devices to perform some computational tasks. With O-RAN, the number of devices is expected to increase significantly. Therefore, O-RAN is expected to handle the resulting traffic more intelligently. One of the key techniques in handling such traffic is the MEC. MEC reduces the congestion over the mobile network by bringing the processes closer to the end user. MEC also contributes to reducing the delay in the 5G networks. The lower latency allows for applications with higher speed requirements.

- Employing ML: ML enables the computer to learn and pick up patterns without the need for explicit programming which perfectly suites a dynamic environment such as O-RAN. Specifically, ML can enhance the resource management and mobility as well as realizing O-RAN as an SON. Many ML models have been already employed in enhancing O-RAN functionalities, such as Q-learning for the resource management and allocation, and Bayesian learning in Multiple-Input Multiple-Output (MIMO) network channel application.

2.2 Related Work

In recent years, dynamic resource allocation has been studied in the context of NFV in general and the RAN in particular. Many algorithms in the literature are concerned with optimizing resource allocation. Peng et al. have proposed an optimization problem for resource allocation with the objective of minimizing power consumption [12]. Shi et al. have investigated implementing a Reinforcement Learning (RL) solution for resource optimization, successfully improving the scalability and utilization of resources in 5G networks [13]. Li et al. have deployed a deep Q-learning for end-to-end resource allocation, considering multi-service and multi-slice scenarios [14]. Considering QoS requirements, Zhou et al. have proposed a multi-agent RL for radio resource management in the context of 5G networks [15]. Bari et al. have formulated this problem into an Integer Linear Programming (ILP) and came up with a heuristic solution for large-scale networks [1]. Yuan et al. have presented a pooling deployment approach that achieved fine-grained management of the resources [16]. Xiao et al. have utilized a policy gradient-based deep RL to autonomously place instances, while capturing the dynamicity of the SFC request and maximizing the QoS [17]. Bunyakitanon et al. have approached the placement problem with the objective of maximizing the availability [18]. The model, based on Q-learning, generalizes well across heterogeneous nodes with varying capabili-

ties. Cohen et al. have investigated VNF geo-placement over different datacenters while minimizing the cost [19]. However, they deal with a static model, not considering the dynamicity of traffic. Elasticity models consider this variability in behavior. Regarding VNF scaling, Arteaga et al. have presented a VNF adaptive scaling using RL, however, the model was SFC specific and does not work with different SFCs [20]. Wang et al. have proposed a dynamic instance provisioning model for enterprise services [17]. The model takes into account the traffic and the server capacity; however, it is reactive; it can result in SLA violation. In [21] and [22], the models developed are proactive in nature; they predict the traffic and scale accordingly. Bilal et al. have formulated the problem into a time-series one to predict resource usage; thus, achieving an elastic NFV [23]. Clayman et al. have focused on developing a dynamic placement model that can handle increasing demand by installing virtual routers, however, it does not have a deprovisioning mechanism [24]. Cloud Providers (CPs) implement elastic solution as additional services. Amazon Auto Scaling Group and Microsoft Azure offer the tenants horizontal scaling solutions [25] [26]. CloudScale and PRESS have employed vertical scaling solutions for allocating and releasing resources, however, in most cases, it does not support changing the resources on-the-fly [27] [28]. Thus, vertical scaling is not recommended by CPs [29]. A latency-aware scaling solution requires a scaling decision to be predicted [30]. Some efforts have employed ML models for

prediction, Mijumbi et al. have employed Graph Neural Network (GNN) to model topological dependencies of VNF Chains [31] [32]. It performed well compared to conventional scaling models; however, the accuracy drops when testing on new data implying a low generalization accuracy.

In the context of 5G O-RAN, resource utilization can be maximized through applying the elasticity techniques of scaling, be it reactive or proactive. Several proposed solutions in the literature focus on elasticity in 5G O-RAN. For instance, Sarrigiannis et al. have proposed an intelligent solution for NFV orchestration consisting of a latency-aware placement, in addition to an online scheduling algorithm that elastically scales VNFs in 5G architecture [33]. However, scaling is done reactively which results in a delay associated with deploying new instances. Gutierrez-Estevez et al. have introduced a reactive ML-based elastic resource management model [34]. CPs such as Amazon have started introducing elastic solutions in the context of 5G networks [25]. However, these solutions are costly and reactive [35].

To the best of our knowledge, a proactive elastic model for resource allocation in the context of 5G O-RAN proposed here has not been studied in the literature. Most existing work related to elasticity is reactive. Moreover, the existing proactive models are generic and do not consider 5G specific requirements. In fact, elasticity in O-RAN environment is still an unexplored area of research. Furthermore,

research efforts have focused primarily on specific areas in the elastic orchestration problem rather than providing an end-to-end solution. For instance, traffic forecasting is an active research area as well as the scaling and placement problems. Many models have been proposed in the literature to address those problems. The solutions proposed vary from optimization, heuristics, to the use of ML. However, to demonstrate the viability of a solution, an end-to-end evaluation has to be carried out. Albeit most elastic models in the literature do not consider network specific requirement, are reactive, or do not generalize well. Hence, this work aims to address this gap by designing and implementing a proactive elastic VNF orchestration policy for 5G O-RAN.

3 Problem Formulation

The service-oriented 5G networks should be able to support a variety of new applications and services with layered performance requirements [36]. Typically, there are 3 families of 5G use cases: enhanced Mobile Broadband (eMBB), massive Machine-Type Communication (mMTC), and ultra-Reliable Low-Latency Communication (uRLLC). The disparity in operational requirements in the different use cases are difficult to be fulfilled by the conventional $one\_size\_fits\_all$ RAN architecture. The introduction of O-RAN allowed the exploitation of intelligent solutions making the network more agile, hence, fulfilling the different operational requirements. As discussed, one use case that employs ML in 5G O-RAN is dynamic resource management. In this Chapter, the problem statement is presented along with the general framework and the workflow sequence.

From a Cloud Provider perspective, a traffic request in O-RAN can be modeled as traffic originating from a source (O-RU) that propagates through the O-RAN SFC consisting of the O-DU, O-CU, and Near-RT RIC (Figure 3.1). We divide

the elastic VNF orchestration problem (VNF-OP) with the objective of minimizing OPEX through scaling VNF instances in a proactive manner into two subproblems, when to scale (traffic prediction) and how to scale (scaling decision and provisioning).

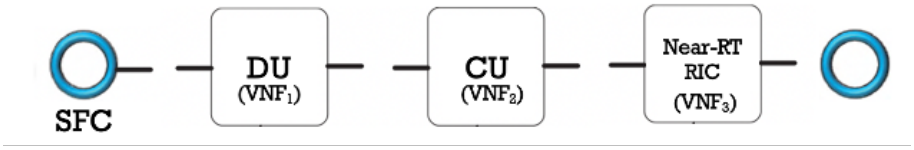

Figure 3.1: Simple O-RAN service function chain (SFC).

The former subproblem is focused on traffic forecasting in an O-RAN context. Traffic forecasting is a crucial part of the elastic orchestration problem. Predicting traffic accurately enables the elastic model to closely match the resources to the traffic ahead of time, thus reducing the operating cost and increasing the utilization as discussed previously. Traffic forecasting can be modeled as a time-series forecasting problem; thus, many suitable ML models have been investigated in the literature.

The latter subproblem is focused on obtaining a scaling decision that captures the dynamicity of the traffic, then according to the decision, instances are provisioned in different nodes such that they can handle the incoming traffic. Since this process is sequential, it is of utmost importance to ensure the accuracy of the forecasting before using the prediction to obtain the scaling decision. Finally, in

accordance with the obtained scaling decision, the O-RAN VNFs are re-deployed in the network such that they can handle the incoming traffic and do not violate the SLAs.

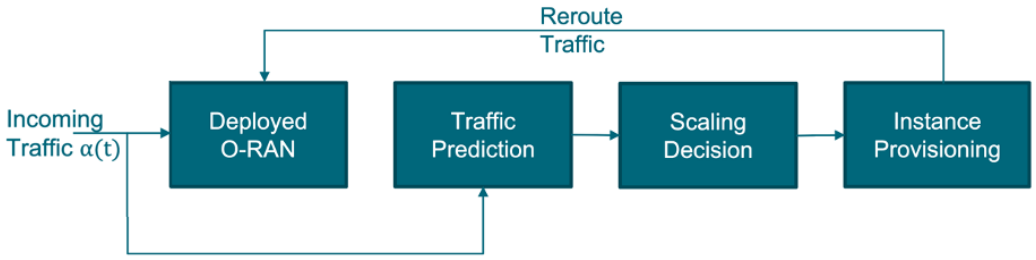

Figure 3.2: Block diagram of the elastic orchestration.

3.1 Framework Roadmap

As mentioned previously, the system has two main parts: traffic forecasting and scaling. In this sense, the system operation at each timestep can be modeled as follows; it starts with the deployed O-RAN units, the incoming traffic is fed into a model that predicts the future traffic, then the predicted traffic is used to extract a suitable scaling decision, and finally, new instances are then provisioned on nodes according to the scaling decision and placed accordingly (Figure 3.2).

Firstly, the initial deployment problem given O-RAN specifications and underlying network infrastructure is addressed. It can be done through an optimization model that computes the optimal placement of the O-RAN VNF such that various

capacity, redundancy, and latency constraints are not violated.

Secondly, traffic forecasting problem is addressed. What makes time-series forecasting problems unique is the dependency between the features, i.e., the model is used to predict a quantity based on a lagged version of the quantity itself. Moreover, prediction confidence diminishes the further the model extrapolates into the future as the correlation starts to decrease. This causes the model to be largely dependent on the quality of the utilized dataset. This dependency puts an emphasis on the model's ability to capture the smallest patterns and pick up on the faintest correlations in the dataset. Therefore, it is inherently more challenging to predict values in a time-series as opposed to a regular regression problem. However, this makes time-series forecasting problems a prime candidate for ML models that tend to perform well in these settings. Traffic forecasting can be formulated as a supervised learning problem where the features input to the model are the past observations (Figure 3.3). Therefore, the prime objective is to use past observations to predict the traffic. Several advanced ML model are suited for traffic prediction. However, their performance is mainly dependent on the utilized data [37]. The amount of deployed O-RAN VNFs is determined by the time-varying traffic; thus, it is crucial to obtain a prediction with minimal error. Furthermore, the prediction depends entirely on the past observations, hence it is of the utmost importance to ensure the quality of the data before going ahead with model implementation.

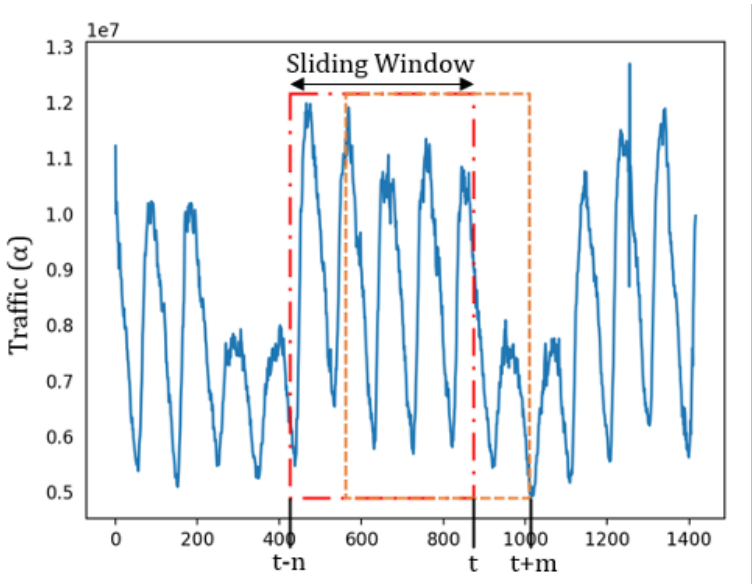

Figure 3.3: Sliding window for time-series prediction.

After acquiring a prediction for the traffic, a scaling decision is needed to tell the system how many instances to add or remove to match the traffic behavior. Obtaining the scaling decision is considered simple task from a theoretical point of view. Conventional models obtain the scaling decision analytically through applying several equations that relates the traffic bandwidth to the processing capacity of the VNF. However, in the context of O-RAN, analytical solutions are hard to apply due to the lack of information and data associated with new generation O-RAN units. Hence, an ML-based solution is proposed to address this issue. Given the traffic prediction from the forecasting stage in addition to the current deployed O-RAN VNF, a scaling decision can be found using ML that reflects the traffic.

Finally, the scaling decision is then passed to the instance provisioning stage that finds the optimal placement for the new instances based on the traffic and the current deployed VNFs. Much like the scaling stage, conventionally the provisioning uses analytical solutions to find which nodes to add instances to and a heuristic model for placing those instances. However, in those models the analytical solution is dependent on the CPU utilization of each VNF instance. In most cases, the CPU utilization is not linear with the traffic and is VNF specific. In the context of O-RAN, the VNF specific information is not available for public discloser. To combat this issue, an RL based model is proposed for provisioning and placement. Using RL with a reward function that captures the requirements of the placement, we can eliminate the need for specifics regarding the different O-RAN VNFs.

The performance evaluation of the methodology is done in two parts. Firstly, at each stage, the implemented models are evaluated based on their predictive capabilities using metrics such as the mean absolute error, in addition to maintaining the SLAs and minimizing the overall latency. Moreover, the models' generalization is then tested on a different dataset to validate the performance. Secondly, the performance of the system as a whole is evaluated through the network utilization for each O-RAN VNF type, in addition to a comparison between the traffic and the number of deployed VNFs.

3.2 Summary

In this Chapter, we have formulated our problem based on the background and related work discussed in the previous Chapter. Furthermore, we have outlined the general framework followed in this work in relation to different O-RAN specific considerations. As discussed, the methodology has four main stages: traffic forecasting, initial deployment, scaling decision, and dynamic placement. In the next Chapter, we go through each stage in more details, highlighting the different design aspects of the models along with some discussion about different challenges and workarounds implemented.

4 Elastic O-RAN VNFs Orchestration

As mentioned previously, the problem is sub divided into traffic prediction and scaling decision. The rationale behind the methodology is as follows; given the traffic arriving at the O-DU propagating through the O-RAN SFC (O-DU, O-CU, Near RIC), an ML model can be utilized to predict the incoming aggregated traffic through the SFC. Using the prediction, a second ML model can be used to decide the number of VNF instances needed for handling the incoming traffic. As with ML models in general, one of the main challenges in the implementation is the feature-label pair for the training.

The traffic prediction problem can be modeled as a supervised time-series problem. The training data can be obtained by applying some preprocessing and analysis on real-world traffic data as discussed in the next section.

As for the scaling problem, obtaining the features-labels pair can prove challenging. Since the objective of the scaling model is to give us the number of VNF instances required to serve the predicted traffic, the feature is the traffic prediction

obtained from the traffic forecasting model and the label is the optimal number of instances for each VNF type. In order to obtain the labels, an ILP algorithm presented in [1] is utilized. The ILP presented tackles the VNF-OP through employing a CPLEX-based model. It takes the traffic requests for a specific timestep and generates an output file with the optimal number and placement of VNF instances that can serve those requests. Using the traffic dataset predicted by the forecasting model, we generate time varying SFC traffic requests based on a realistic dynamic NFV environment for each traffic instance. Afterwards, at each traffic instance, the associated traffic requests are input to the ILP model - adapted to our O-RAN use case - generating an output file. Finally, we extract relevant information, i.e., number of VNF instances per type, placement, etc. from the generated output files. The extracted information then serves as labels for the training set.

Following the acquisition of the scaling decision, VNF instances in the network has to be redeployed in accordance with the decision; either to scale up or down. The redeployment is done through a dynamic placement scheme that takes into account the dynamic nature of the network along with O-RAN specific constraints. Conventional optimization models can solve the dynamic placement problem; however, the problem has to be reformulated at each timestep since the placement is changed after each scaling decision. Moreover, heuristic solutions attempt to address this limitation, however the results are sub-optimal placement, since it

trades-off speed with accuracy. The next viable candidate would be an ML model. The changing environment along with the lack of enough training dataset makes it challenging for the ML model to generalize and achieve a good performance. Hence, we propose an RL agent for the dynamic placement problem to address the aforementioned concerns. The advantage of RL agents is that they do not require labeled training dataset similar to that of other ML models. Alternatively, the agent is supposed to learn from its past experiences in the environment. However, setting up the environment and defining a reward function can be quite challenging depending on the problem.

Keeping the aforementioned challenges in mind, we start off by introducing the general setting of our O-RAN VNF-OP as follows [1]:

(1) The physical network is presented as an undirected graph $G = (N, L)$, where N and L represent the set of nodes and links respectively. Since the O-RAN components are virtualized, they are treated as VNFs that can be deployed on commodity COTS servers located in the network [38]. The set of S represents those servers. The set of resources available for each server (RAM, CPU, HDD) is denoted by R. The resource capacity for server s is represented by $c_s^r \in \mathbb{R}^+, r \in R$. The propagation delay along with the bandwidth capacity of a link are modeled as l_d and $l_b w$ respectively.

(2) The network function types are represented by a set T, where each VNF

type $t \in T$ requires a specific amount of resources $u_t^r \in \mathbb{R}$. Additionally, each VNF type has a processing capacity and delay ($p_t, p_d \in \mathbb{R}$).

(3) The traffic demand in the network can be modeled as a request originating from a source node to a destination node, along with the required bandwidth, tolerated delay, and the VNF sequence requested. Hence, it can be represented as a tuple $tr = <tr_{src}, tr_{dst}, tr_{bw}, tr_d, tr_{seq}>$.

Given a timestamp t_0, the task of the model is to predict a scaling decision representing the number of VNF instances that have to be provisioned or deprovisioned to handle the incoming traffic at timestamp t_p, where p is the prediction horizon. This problem can be divided into four sub-problems; predicting incoming traffic, calculating the number of VNF required based on the predicted traffic, identifying the scaling decision, and finally placing new VNF instances based on the scaling decision. After that, the new number and placement of the VNFs can be relayed to Service Management and Orchestration (SMO) component for deployment. In this section, the methodology for addressing each sub-problem is discussed. Moreover, the model used for each part along with the dataset utilized are discussed.

4.1 Traffic Forecasting

Traffic forecasting is a crucial part of the elastic orchestration problem. Predicting traffic accurately enables the elastic model to closely match the resources to

the traffic demands ahead of time, thus reducing the operating cost and increasing the utilization as discussed previously. Traffic forecasting can be modeled as a time-series forecasting problem. Time-series forecasting can be formulated as a supervised learning problem where the features input to the model are the past observations (Figure 4.1). Therefore, the prime objective is to use past observations to predict the traffic. However, what makes time-series forecasting problems unique is the dependency between the features, i.e., the model is used to predict a quantity based on a lagged version of the quantity itself. Moreover, prediction confidence diminishes as the model extrapolates further into the future. Therefore, it is inherently more challenging to predict values in a time-series as opposed to a regular regression problem. The forecasting problem can be formulated as follows:

$$\alpha^*(t+m) = f(\alpha(t), \alpha(t-1), \alpha(t-2), ..., \alpha(t-n)) \tag{4.1}$$

where α^* is the predicted traffic m timesteps in the future, α is the actual observed traffic, and n is the size of the observation window.

Since the prediction depends entirely on the past observations, it is of the utmost importance to ensure the quality of the data before going ahead with model implementation. The quality of the data is dictated by two main factors: the time length of the observations and the predictability of the series itself. Firstly, the length of the available series matters greatly in training the model as the available number of samples must be greater than the number of model parameters [37]. Hence, the

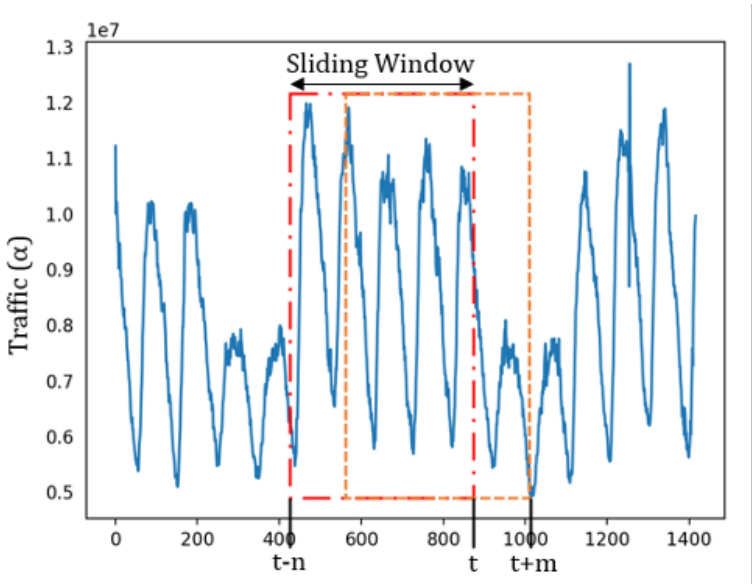

Figure 4.1: Sliding window for time-series prediction.

more complex the model is, the more samples it needs for training. Secondly, the predictability of the series quantifies the regularity and the predictability of the fluctuations in the series; the more regular and repeated patterns the series has, the easier it is to forecast. The predictability depends on the temporal granularity of the observations; temporal aggregation significantly degrades the predictability of the series. The predictability of the series can be ensured through applying some time-series analysis techniques. By analyzing the components of the series through decomposition, predictability can be inferred. A time-series can be decomposed into 3 main components: *i*) a trend component, representing the traffic growth over time, *ii*) a seasonal component, representing the cyclical behavior of the se-

ries, and *iii*) a remainder component, representing abnormalities due to sudden events (such as cell outage in the case of RAN). For a series to be predictable, it must have a high level of stationarity, meaning the statistical properties of the series (mean, variance, and autocorrelation) are not a function of time. Moreover, the parameters for the model can be deduced for the statistical properties of the time-series. For instance, the size of the observation window is inferred from the autocorrelation function. Also, depending on the model used some transformations maybe needed to ensure stationarity or to ease the prediction process.

The traffic forecasting problem is modeled as a supervised learning problem where the task is to predict the upcoming traffic according to the features extracted from a sliding window. In other words, the model learns the pattern between the provided features and the labeled predictions. Hence, the first step is to acquire the traffic data, apply some preprocessing to extract the features from the sliding window, then feed the features to the model. By acquiring a prediction from the model, the elastic orchestrator can use it to compute the number of O-RAN VNFs required to handle the traffic. By comparing the required VNFs with the current provisioned one, a scaling decision can be reached. Finally, based on the scaling decision, new instances are placed, and new placement is relayed to the Service Management and Orchestration (SMO) component for deployment. In this section, we discuss the ML models proposed for the traffic forecasting problem along with

model-specific transformations and preprocessing applied to the dataset to ensure stationarity and compatibility with the model.

After considering the available models, we choose two ML models that are suitable for our use case; Auto Regressive Integrated Moving Average (ARIMA) as a classical statistical model and Long-Short Term Memory (LSTM) as a state-of-the-art algorithm. We chose these models for their superior performance in time-series problems [39]. A brief description of the models and their setup is discussed below:

4.1.1 ARIMA

It is a class of statistical models for analyzing and forecasting time-series data. It can represent a given time-series based on its own past values, i.e., its own lags and the lagged forecast errors. ARIMA process can be described as a function $ARIMA(p, d, q)$. The parameters p, d, and q describe the non-seasonal part of the time-series. Namely, p is the order of auto-regression, d is the level of difference, and q is the order of moving average. Mathematically, the ARIMA model can be represented as follows:

$$\alpha_t = A + B_1\alpha_{t-1} + B_2\alpha_{t-2} + ... + B_p\alpha_{t-p}\epsilon_t + \phi_1\epsilon_{t-1} + \phi_2\epsilon_{t-2} + ... + \phi_q\epsilon_{t-q} \quad (4.2)$$

In designing the ARIMA model, the objective is to determine the values of p, d, and q. Firstly, the value of d dictates the number of times the series is differenced. The differencing is applied to the series to eliminate the trend component which in

turn ensures stationarity. To determine how many times the series is differenced, we employ the Augmented Dickey Fuller (ADF) test for stationarity. To determine the level of difference, we check the stationarity after each differencing using the ADF test until the series is stationary, i.e., the p-value of the ADF test is less than 0.05. Hence, d represents the number of times the series is differenced until it reaches stationarity. Secondly, we determined the value of p through the Partial Autocorrelation Function (PACF). By plotting the PACF, the number of lags over the significance line dictates the value of p. Lastly, in a similar manner to p, we deduce the value of q from the plot of the Autocorrelation Function (ACF). Moreover, prior to inputting the series into the model, we apply some preprocessing to ensure stationarity and compatibility with the model. As discussed previously, the trend is eliminated through differencing, however, the seasonality is not. Therefore, we eliminate it beforehand through a nonlinear transformation that guarantees the statistical properties of the series (mean, variance, and autocorrelation) are time independent. Afterward, we normalize the series and input it to the model.

4.1.2 LSTM

It is a special kind of Recurrent Neural Networks (RNN), capable of learning long-term dependencies. LSTMs have an advantage over conventional RNNs in that they are designed to avoid the long-term dependency problem. Another advantage

LSTMs have over RNNs is their ability to deal with the vanishing gradient problem during training. All this makes LSTM networks well-suited to make predictions based on time-series data.

Prior to inputting the series to the model, we apply some preprocessing in terms of sliding window and feature extraction. The preprocessing reshapes the dataset from a sequential dataset to a supervised learning dataset with the present and past N values as features and the value at $t + M$ as label. We apply the preprocessing stage for two reasons; to transform the series into a format acceptable by the model, and to make it easier for the model to pick up on patterns in the series; thus, contributing to a more accurate prediction. Since the series becomes less correlated with itself as the lag value increases, it makes it tricky to choose the appropriate values for both $N\&M$. For instance, when choosing the value of N, the more past values included in making a prediction, the more accurate it becomes since more information is used to make the prediction. However, since the ACF declines as the lag increases, including more past values does not always contribute to the prediction accuracy. On the contrary, it just increases the complexity of the model which, in turn, makes it harder for the model to converge. Therefore, we derive the value of N from the ACF; we include past values up to the last significant autocorrelation value. As for M, to ensure accuracy of prediction and reduce the complexity of the model, we choose to only predict one timestamp.

Moreover, we include two more features to capture the periodicity in the series and the effects of seasonality on the prediction; Day_of_week to make a distinction between weekday versus weekend traffic, and $Hour\_of\_day$ to differentiate between daytime as opposed to nighttime traffic. We employed the Granger causality test to ensure the usability of those features in predicting our time-series. Finally, we rescale and normalize the dataset to fit the criteria of the neural network. In terms of the model parameters, we adopted some of the best practices in literature along with some tuning to improve performance when choosing number of layers, number of LSTM units in each layer, learning rate, optimizer, activation function, weights initializer, and loss function.

4.2 Initial Deployment

Before we can scale instances according to the predicted traffic, there has to be an already deployed O-RAN instances. Then according to the prediction, we can come up with a scaling decision to modify the number of active VNF instances to match the traffic. Initial deployment problem can be tackled given only the O-RAN specifications and the underlying network infrastructure since it does not account for the traffic dynamicity. It can be done through an Integer Linear Programming (ILP) model that computes the optimal placement of the O-RAN VNFs such that no constraints are being violated, thus maintaining the SLAs (Figure 4.2). One

of the major concerns when it comes to VNF placement in an O-RAN environment is the reconciliation of the resource utilization (resource fragmentation) and minimizing the delay. Service providers often have to deal with this problem in order to minimize their operating costs while maintaining the SLA. Furthermore, the separation of the edge cloud and the regional cloud in the O-RAN architecture adds another layer of complexity. The edge cloud provides a significantly less propagation delay, however, it has a limited amount of resources and over-provisioning instances can cause underutilized resources infrastructure. On the other hand, the regional Cloud provides a large amount of physical resources and a higher level of reliability, but it adds a delay overhead that can affect the SLA. Moreover, when deploying an SFC, some VNFs are constrained to either regional or edge Cloud depending on the functional split option defined by the *O-RAN Alliance*. As discussed in Chapter 2, O-RAN architecture split the RAN functionality between the O-CU, O-DU, and O-RU. There are eight functional splits proposed and studied in the literature differing in fronthaul capacity to transmit radio signals and operating cost [11]. The options range from a separated user and control plane option 1 to a fully centralized option 8, with each option having some advantages and drawbacks. Thanks to a huge research effort in the literature, it turns out that the split family 7.x offer the best balance between the intercell cooperation and the complexity of the O-RU. Furthermore, since the *O-RAN alliance* has more specifically considered

option 7.2, we consider this option throughout this work [40]. According to option 7.2, the Near RIC should be hosted in the regional Cloud while the O-CU and O-DU are hosted on the edge Cloud [41].

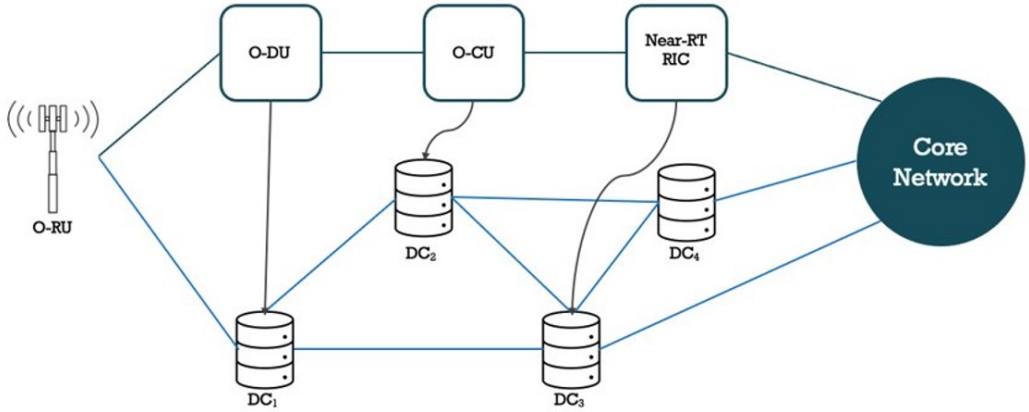

Figure 4.2: SFC placement.

Keeping the aforementioned points in mind along with the importance of the delay between the VNFs in an O-RAN SFC, the objective is to find the optimal placement of VNF instances. Considering the O-RAN requirements, the optimal placement should provide minimal delay in addition to abiding by the capacity, latency, and regional/edge Cloud constraints. Moreover, the model is topology specific, where the topology we have considered throughout this work is the Internet2 network topology [42]. The topology consists of 12 nodes and 15 links along with capacities for the links and the associated delays, in addition to the available physical resources at each node (Figure 4.3). Appendix A shows a sample

implementation of the optimization model on IBM™ CPLEX.

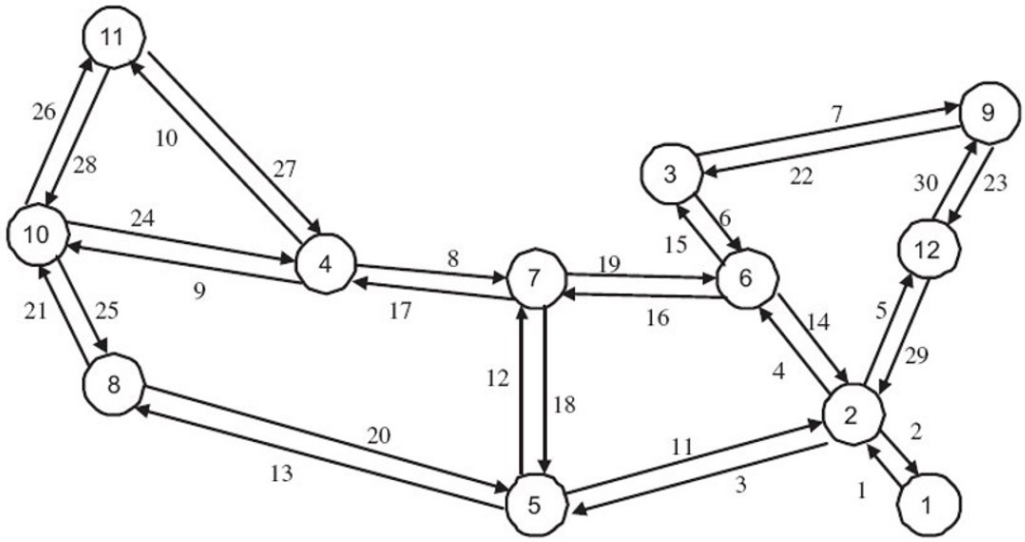

Figure 4.3: Internet2 network topology.

Firstly, we define the decision variable x_{ij} to map the VNF instances onto the physical servers:

$$x_{ij} = \begin{cases} 1, & \text{if VNF } i \text{ is allocated in server } j, \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

Next, we define the variable w_{ik} representing the dependency between the VNF instances, i.e., whether or not two instances are communicating as part of an SFC:

$$w_{ik} = \begin{cases} 1, & \text{if VNF } i \text{ depends on VNF } k, \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

Additionally, we define two more variables r_j and e_j to represent whether a server is part of a regional or edge cloud respectively:

$$r_j = \begin{cases} 1, & \text{if server } j \text{ is a Regional Cloud server,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

$$e_j = \begin{cases} 1, & \text{if server } j \text{ is an Edge Cloud server,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.6}$$

Given the set of resources C available by each server s, the resources requirement U required by each VNF type t, and the latency matrix D^s_{ij}:

$$C = [c_1, c_2, \cdots, c_n] \tag{4.7}$$

$$U = [u_1, u_2, \cdots, u_m] \tag{4.8}$$

$$D^s_{jz} = \begin{bmatrix} d_{11} & d_{12} & d_{13} & \dots & d_{1s} \\ d_{21} & d_{22} & d_{23} & \dots & d_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{s1} & d_{s2} & d_{s3} & \dots & d_{ss} \end{bmatrix} \tag{4.9}$$

We can derive the set of latencies D^t_{ik} representing the maximum allowed latency

between VNF instances in an SFC:

$$D_{ik}^{t} = \begin{bmatrix} d_{11} & d_{12} & d_{13} & \dots & d_{1t} \\ d_{21} & d_{22} & d_{23} & \dots & d_{2t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{t1} & d_{t2} & d_{t3} & \dots & d_{tt} \end{bmatrix} \tag{4.10}$$

Finally, to map the dependency between VNF instances w_{ik}, we define the variable wl_{ik}^{jz} representing whether or not link l_{jz} is utilized by the w_{ik} connection:

$$wl_{ik}^{jz} = \begin{cases} D_{jz}, & \text{if connection } w_{ik} \text{ uses link } l_{jz}, \qquad \forall\ j, z \in [1, m] \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

Now, to ensure the capacity of each server is not exceeded by the deployed VNFs, it is represented by the following constraint:

$$\sum_{i=1}^{n} x_{ij}.c_i \leq u_j, \qquad \forall\ j \in [1, m] \tag{4.12}$$

Additionally, to enforce the distinction between the Regional and Edge Cloud, the following constraints are applied:

$$\sum_{j=1}^{m} x_{ij}.r_j = 0, \qquad \forall\ i \in eVNF \tag{4.13}$$

$$\sum_{j=1}^{m} x_{ij}.e_j = 0, \qquad \forall\ i \in rVNF \tag{4.14}$$

Where $eVNF$ and $rVNF$ represent the set of VNFs that are constrained to the Edge and Regional Cloud respectively.

Next, to guarantee the maximum allowed latency is not violated, the following constraint applies:

$$\sum_{z=1}^{m}\sum_{j=1}^{m} w_{ik}.x_{ij}.x_{kz}.D^{s}_{jz} \leq D^{t}_{ik}, \qquad \forall\ i,k \in [1,n] \tag{4.15}$$

Finally, we need to ensure that every VNF has been placed exactly once:

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad \forall\ i \in [1,n] \tag{4.16}$$

The final objective here is to find the optimal placement for the VNF instances that minimizes the overall latency between the instances subjected to all the aforementioned constraints, hence realizing the SLAs. The objective function can be formulated as:

$$\begin{aligned} &\min_{X} \quad D, \\ &\text{where} \quad D = \textstyle\sum_{z=1}^{m}\sum_{k=1}^{n}\sum_{j=1}^{m}\sum_{i=1}^{n} wl^{jz}_{ik} \end{aligned} \tag{4.17}$$

4.3 Scaling Decision

Theoretically speaking, obtaining the scaling should be a simple task. Conventional models obtain the scaling decision analytically through applying several equations that relates the traffic bandwidth to the processing capacity of the VNF. However, in the context of O-RAN, these analytical solutions are hard to apply due to the lack of information and data associated with the new generation O-RAN units. Hence, we propose an ML-based solution to address this issue. Given the traffic prediction from the forecasting stage, we can obtain a scaling decision using ML.

As mentioned previously, we need to generate a training dataset to train the ML model. To generate this dataset, we utilize the ILP solver proposed in [1]. The ILP takes a set of traffic requests as an input, and provides the optimal number and placement for the VNF instances in accordance with the traffic requests for a specific traffic instance. Through adapting the ILP to our use case, then running the solver for each traffic request, we can generate a features/label dataset with the traffic as the feature and the optimal number of VNFs per type as a label.

The general setup for the ILP solver includes the network topology, the VNF catalog, and the traffic matrix, in addition to parameters controlling the network load and the composition of the SFCs. In this work, for the traffic topology, we utilized the Internet2 network topology with 12 nodes and 15 links [42]. The server data used are the ones provided in [1]. Since the SFC use case scenario in this work is O-RAN focused, this topology was well-suited since it represents a Wide Area Network (WAN) topology; hence, covering the perspective of service providers who can instantiate VNF instances at different locations in a regional or a global cloud. The traffic matrices used are the same as the ones used for training the traffic forecasting model. For the VNF catalog, we utilized the O-RAN data published in [43] and [11]. It contains data regarding the resource consumption and resulting bandwidth capacity for different O-RAN VNFs. We generate the training data for the ML model in two stages; we generate the SFC requests for each traffic instance,

and we run the ILP for each traffic instance to generate the optimal number of VNFs. After generating the dataset, we train a Neural Network on the data to capture the relationship between the traffic and the number of VNFs required.

4.3.1 SFC Request Generation

Given the aggregated traffic matrix, the first step is to generate the SFC requests to be input to the ILP solver. Through defining a peak arrival rate and an average duration for the SFC requests, it is possible for us to control the number of active requests propagating through the SFC at each time instance. Moreover, the ILP model allows for defining different SFC lengths and compositions with different occurrence probability. However, for the purposes of this work, we define the SFC as the O-RAN SFC discussed with a 100% occurrence probability. The variation of the arrivals is dictated by the traffic volume and the distribution used to generate the inter-arrival rate. The time-varying traffic volume is used to modulate the arrival rate. The arrival rate is set as the product of the peak rate and the normalized traffic volume. Additionally, the distribution of the duration and the required bandwidth also has an effect on the arrival rate variation. After modulating the arrival rate, we use the time-varying arrival time to generate SFC requests. Each SFC request consists of a tuple $tr = <tr_{src}, tr_{dst}, tr_{bw}, tr_{d}, tr_{seq}>$, representing the arrival time, duration, source and destination nodes, requested bandwidth, and the

VNF sequence.

4.3.2 Training Data Generation

Given the set of SFC requests, the ILP solver takes each request and produces the optimal number and placement of the VNFs required after each arrival. The ILP outputs a set of active VNFs in the pseudo-topology used in the ILP. We then extract the number of active VNF instances per type from the output file. The number of active instances along with the traffic trace represents the feature/label dataset needed for training the ML model.

Since the forecasting is done at the traffic prediction stage, the relationship between the SFC requests and number of VNF instances can somewhat be approximated as a linear relationship. Hence, we leverage a Multi-Level Perceptron (MLP) to model the relationship. The implemented MLP takes the traffic as an input, and outputs the number of VNF instances required per type. Since the scaling decision prediction is done per VNF type, we implement three models corresponding to each VNF type (O-DU, O-CU, Near RIC). Prior to inputting the data to the MLP, we rescale and normalize the dataset to fit the criteria of the neural network. In terms of the model parameters, we adopted some of the best practices in literature along with some tuning to improve performance when choosing number of hidden layers, number of nodes in each layer, clip value, learning rate, optimizer, activation func-

tion, weights initializer, and loss function. The scaling MLP generates the number of VNF instances per type in accordance with the traffic prediction. Through comparing the deployed VNF instances (deployed using the initial deployment model discussed in Section 4.2) with the optimal number of instances generated by the scaling ML models, we can obtain a scaling decision. We represent the scaling decision by the vector $SD = [\Delta\text{O-DU}, \Delta\text{O-CU}, \Delta\text{Near RIC}]$.

4.4 Dynamic Placement

After obtaining the scaling decision, the next step is to provision and/or deprovision VNF instances according to the acquired scaling decision. Given the scaling decision vector, the new instances have to be deployed in the available servers in the network. Nonetheless, the dynamic placement problem can prove quite challenging. The dynamic placement problem and the initial placement problem discussed previously differ mainly in two aspects; the dynamicity of the network, i.e., the changing environment in terms of VNF deployment, and the active request in the network that does not allow for drastic changes in placement. The initial placement problem discussed is a static problem, i.e., the model is to deploy instances on servers where no instances are deployed. However, the dynamic placement problem aims at addressing the placement problem during network operation. Instances are already deployed on various servers with active requests running throughout the

network. Hence, the model must deploy new instances cognizant of the already deployed ones in the network. Consequently, the model must recognize that servers are partially occupied, connections are stablished, and requests are running. Moreover, the active requests in the network limit the number of instances the model can redeploy to prevent rerouting a significant number of requests. Furthermore, after each deployment the problem is reformulated since the new placement changed the mapping between the virtual nodes and physical servers. Hence, the model needs to adapt to the changing environment while minimizing the number of instances redeployed. As instances are being placed, the model should be mindful of the capacity, latency, and redundancy constraints. Additionally, the active requests in the network hinder the model's ability to change instances that are part of the same SFC. Moreover, when adding an instance to a VNF node it acts as an image to the already deployed instances at that node, hence, it should be able to communicate with all the instances in the same SFC. All these factors add layers of complexity to the problem of initial deployment.

An ILP model is capable of solving the placement problem, however the problem has to be reformulated at each time instance. Albeit ILP solutions can give an optimal placement, it is highly inefficient in this case due to their static nature, the need for problem reformulation after each deployment, and the long execution time it takes at each timestep to solve the problem. Hence, it is not suitable for an au-

tonomous real-time use case such as O-RAN. Heuristic solutions attempt to address this limitation; however, the results are sub-optimal placement, since it trades off speed with accuracy, in addition to their lack of scalability in terms of how many instances it can support. Therefore, as discussed in Chapter 2, various research efforts are directed toward exploiting ML solutions in addressing the VNF placement problem. However, due to the changing placement environment, the lack of enough training dataset, and the difficulty of the model to generalize its performance to different placement scenarios, conventional supervised learning models struggle in achieving superior performance. To this end, we propose a Reinforcement Learning (RL) agent that can handle the limitations of the supervised learning techniques. Similar to the conventional ML models, RL agents have a short execution time and are highly scalable and dynamic, in addition to maintaining a near-optimal performance after training. Hence, they address the aforementioned limitations of the ILP and heuristic solutions. Moreover, unlike conventional ML models, RL agents are self-trained and do not require a huge training dataset. Furthermore, RL agents aim to maximize the reward at the end of each episode; thus, they tend to perform better in complex problems compared to conventional ML models. Through defining a proper reward function, the RL agent can learn to prioritize actions that help maintaining the SLAs which conventional ML models struggle to capture. Hence, all these factors make RL agents a prime candidate in tackling the dynamic place-

ment problem. However, setting up the environment and designing a proper reward function are quite challenging and pivotal to the agent's performance. Since the scaling decision just shows the number of VNF instances per type, there are two pieces of information missing for the agent to place instances; which node should the VNF instance mirror, and the server at which the VNF is placed. Afterwards, the model can start placing instances to train the agent.

4.4.1 Physical Network Transformation

We adopt the network transformation proposed in [1] as a preprocessing for the network topology. As defined previously, the set of available servers in the network is represented by S. The transformation of the physical network aims to generate an augmented pseudo-network that reduces the complexity involved in solving the VNF-OP. The transformation process is performed by enumerating the VNF. First, the maximum possible number of each VNF type that can be deployed in all servers is calculated. Then the VNFs are enumerated from 1 to M. The set of enumerated VNFs is called pseudo-VNFs. Hence, each pseudo-VNF $m \in M$ is attached to a server $s \in S$. In a nutshell, if a server s has 20 cores available, and the CPU requirement of VNF_1 and VNF_2 are 4 and 6 cores, respectively, the number of pseudo-VNFs that can be deployed are 5 and 3 of VNF_1 and VNF_2 respectively. Furthermore, to map the pseudo-VNFs onto the physical servers, we

define a mapping function δ where:

$$\delta(m) = s, \qquad \text{if pseudo-VNF } m \in M \text{ is in server } s \in S \tag{4.18}$$

This transformation reduces the complexity of the placement problem in two ways; the model does not need to calculate the maximum number of instances and whether the placement violates it or not, and the model output is simplified to a binary vector Y representing which pseudo-VNF is active, i.e., the VNF is placed in the associated server. We first define y_m indicating whether a pseudo-VNF is active or not:

$$y_m = \begin{cases} 1, & \text{if pseudo-VNF } m \in M \text{ is active,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.19}$$

$$Y = [y_1, y_2, \cdots, y_m] \tag{4.20}$$

Hence, through adopting the physical transformation, the objective becomes designing a model that outputs the vector Y, which represents a placement of all VNF instances, subjected to the various constraints discussed in Section 4.2 and in accordance with the scaling decision.

4.4.2 VNF Node Placement

Following the acquisition of the scaling decision, we know the number of VNF instances to be provisioned per type. However, that is not enough information for

the model to place the instances. The model needs to determine which node the instance is mirroring. In other words, the model should determine which SFC the new instances should be provisioned on.

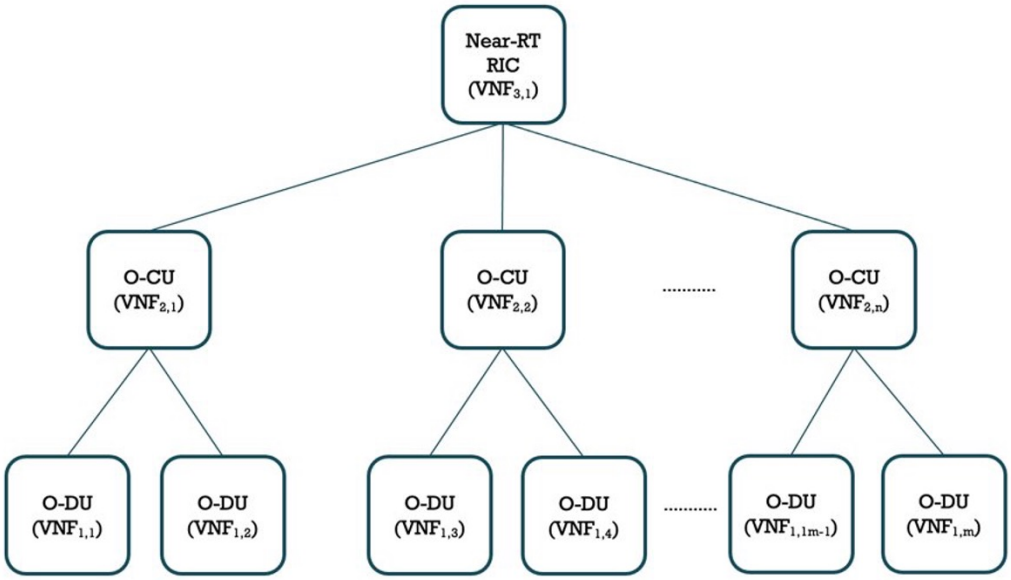

Figure 4.4: Deployed O-RAN SFCs.

As shown in Figure 4.4, the deployed O-RAN consists of nodes for each VNF type. Whenever a new instance is provisioned, it has to be associated with a specific node. Hence, it will inherit all the connections that the node has, thus, becoming part of the SFC and sharing incoming requests. For the model to choose a node, we employ the number of active requests at each node to make a decision. The rationale being that the node with the highest number of active requests signifies that it is overloaded. Furthermore, since the inter-arrival and number of requests

are randomly sampled from a distribution, the nodes receiving a high number of request changes over time, hence, simulating a real-life dynamic network environment. Figure 4.5 illustrates the process of choosing a node to provision instances on. As the number of requests increases on $VNF_{2,1}$, it is chosen for provisioning and instance $VNF_{2,n+1}$ is provisioned. The number of active requests at each node is extracted from the ILP solver discussed in Section 4.3.

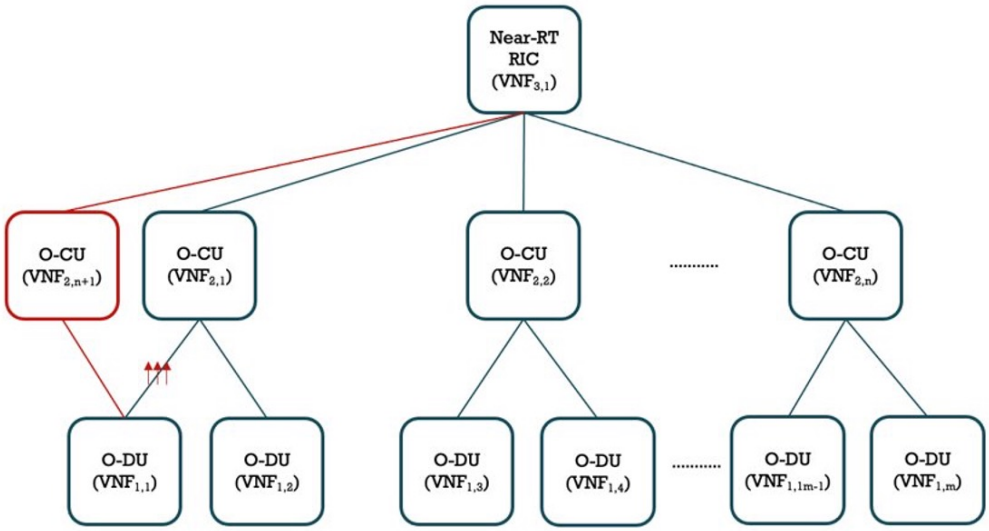

Figure 4.5: Instance provisioning in deployed O-RAN.

4.4.3 RL Model

RL has been the focus of many recent research efforts. As discussed in Chapter 2, RL has been used to address the limitations of conventional optimization and heuristic solutions in the context of resource management. The lack of adaptability

of optimization models to the dynamic network environment and the sub-optimal solution provided by heuristic solutions make ML the next viable solution. However, the difficulty of obtaining enough training dataset for placement that can train the models in different network conditions, and the difficulty to obtain a generalized model that can perform under changing network conditions have illustrated the unsuitability of supervised learning solutions. To this end, RL solutions have been the focus of recent efforts.

RL differs from traditional ML models in two main ways (Figure 4.6). Firstly, RL models are self-trained and do not require huge training dataset in the same way traditional ML models do. Secondly, during training, the objective of RL agents is to maximize the reward or the value function, contrary to other ML models where the objective is to minimize the error between model's prediction and actual value.

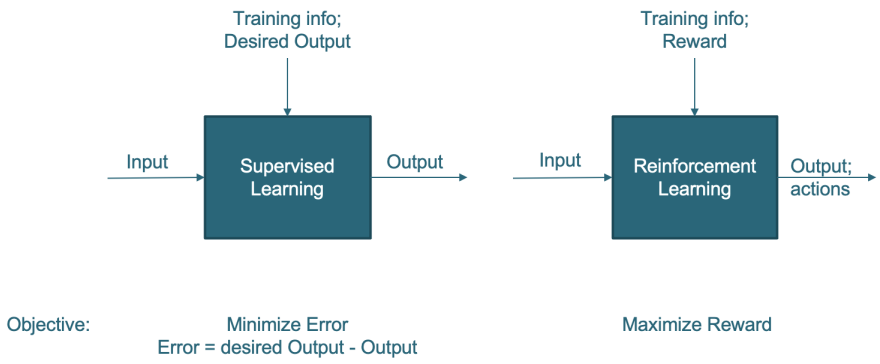

Figure 4.6: Reinforcement learning vs. supervised learning.

As shown in Figure 4.7, there are five main components to the RL model: agent, environment, state, reward, and action. At each point in time, the agent observes the environment to determine the current state S_t, then according to the state, the agent determines a specific action A_t to influence the environment. Finally, the environment is adjusted according to the action, resulting in a new state S_{t+1} and a reward R_{t+1}. Then the cycle repeats until the end of the training episode.

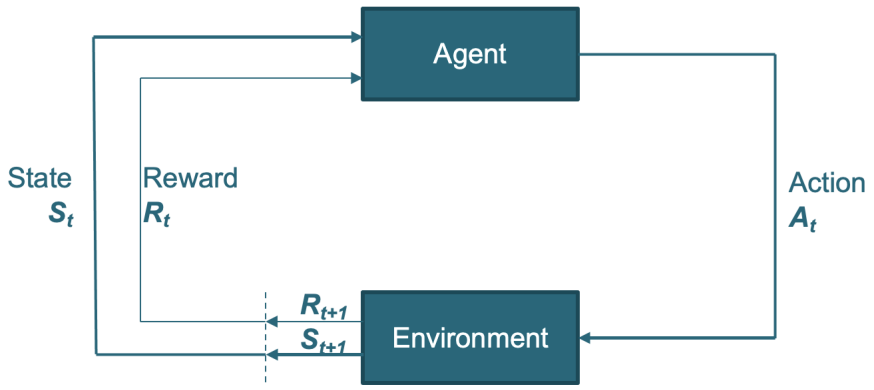

Figure 4.7: Block diagram of reinforcement learning.

In our case, the environment represents the deployed O-RAN network. The states observed by the agent corresponds to the vector Y showing which pseudo-VNFs are active. Finally, the action produced by the agent is a new vector Y corresponding to a new set of active pseudo-VNFs according to the scaling decision.

There are three main ways to implement RL (Figure 4.8); Value-Based, Policy-Based, and Model-Based. Each approach differs mainly in its objective during the

training. In Value-Based RL, the aim is to maximize the value function of the state; thus, the agent prioritizes the future reward to the current one. In Model-Based RL, the agent creates a model of the environment to help generate a suitable action for each environment; hence, the aim is to generate as accurate a model as possible. Finally, in Policy-Based RL, the aim is to maximize the reward at each point in time. Each method has its advantages, however, the problem at hand dictates which approach is more suitable.

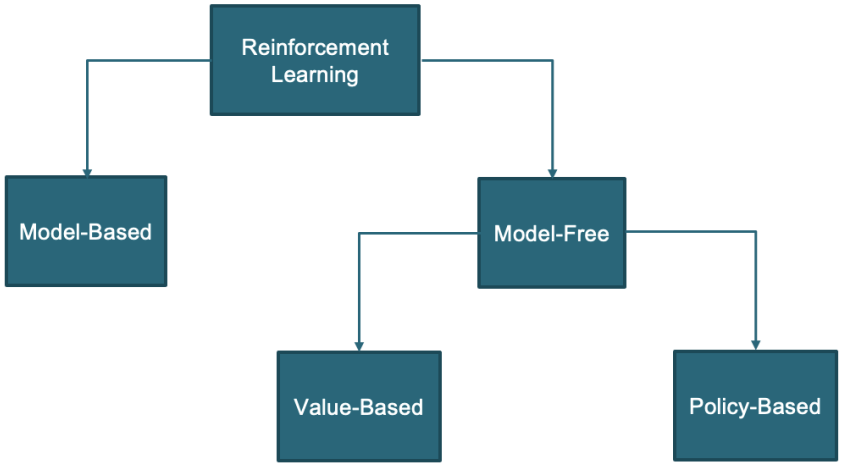

Figure 4.8: Approaches for reinforcement learning.

Regarding the use case discussed in this work, Model-Based RL would not be suitable. The network environment is entirely dependent on the change in demand as VNF instances are provisioned and deprovisioned dynamically according to the incoming demand. Hence, trying to model the environment would not be cost-

effective as there are too many scenarios to consider and will only add a layer of complexity to the problem. Therefore, Model-Free approaches are more suitable.

Value-Based RL learns an action-value function approximation that is then used to optimize for the policy. This makes the Value-Based RL less stable, less reliable, and suffer from poor convergence. Furthermore, Value-Based RL can result in instantaneous violation of constraints, hence failure to fulfill SLAs, since it updates the policy according to the future reward, i.e., violating current constraints in favor of future reward. Policy-Based RL directly optimizes for the policy; hence it tends to be more stable and less prone to failure. Policy-Based would prioritize maximizing current reward; thus, fulfilling the constraints at each point in time. Therefore, in the interest of reducing the complexity of the problem and maintaining the level of performance, the RL approach most suitable for our use case is Policy-Based RL.

4.4.3.1 Environment Setup

The environment is setup based on the physical network transformation discussed. The environment consists of two parts; the set of pseudo-VNFs the network can support, and the scaling decision along with the distribution of the active requests derived from the previous stages. The set of pseudo-VNFs is calculated based on the network topology and the resource requirement for each O-RAN VNF. Each

pseudo-VNF is associated with a server $s \in S$ as discussed. Hence, the environment is topology specific. The number of pseudo-VNF for each type can be calculated as follows:

$$pVNF^t = \sum_{s=1}^{m} \lfloor \frac{c_s^r}{u_t^r} \rfloor, \qquad \forall\, t \in T \;\&\; r \in R \tag{4.21}$$

The pseudo-VNFs are represented in the environment by a vector Y'. Y' has a length equal to the number of pseudo-VNFs available in the network. Each element represents a pseudo-VNF and has a value ranging from 0 to n; if its value is equal to 0, it is inactive, otherwise, it is active. In the case of active pseudo-VNF, the value represents which node that pseudo-VNF is attached to.

The second part of the environment setup is the scaling decision and the active request. They are represented as two vectors; SD represents how many instances are to be added for each VNF type, and Req with a length equal to the number of nodes shows how many requests are active in each node. These two vectors are updated at each iteration according to the results from the previous stages.

At each iteration, the agent observes the environment, i.e., reads the values of the three vectors, then it analytically assigns the instances to be added to nodes in the network. This assignment is done proportionally to the active requests. In case of provisioning instances, the nodes with the highest number of active requests are given priority in instance provisioning, and in case of deprovisiong, the nodes with the lowest number of active requests are given priority for deprovisioning.

This results in a vector $Prov$ that is essentially the combination between the two vectors SD and Req. Finally, the agent's neural network takes both $Prov$ and Y' as an input, and outputs a new vector Y'_{t+1} as an action which represents the new placement. The environment takes the generated action and adjusts accordingly in addition to calculating the reward the agent gets from this action.

4.4.3.2 Reward Function

We formulat the reward function based on the initial placement model discussed in Section 4.2. After the agent sends the action to the environment, the environment is adjusted, and the reward function is triggered to assess the action and calculate the reward. The reward is calculated through summing the rewards resulting from different requirements the placement has to fulfill.

Firstly, the placement should not violate the constraints imposed by O-RAN requirements, hence preserving the SLA. These constraints consist of the capacity, latency, and redundancy constraints. The capacity of each server should not be exceeded and the latency between the placed VNFs should be below the tolerated amount. Moreover, the agent should ensure that each VNF is placed exactly once. To realize these constraints, we formulate the rewards R_1, R_2, and R_3 representing

the capacity, latency, and redundancy constraints respectively.

$$R_1 = \begin{cases} +a, & \text{if } \sum_{i=1}^{n} x_{ij}.u_i \leq c_j, \qquad \forall\ j \in [1, m] \\ -a, & \text{otherwise.} \end{cases} \tag{4.22}$$

$$R_2 = \begin{cases} +b, & \text{if } \sum_{z=1}^{m} \sum_{j=1}^{m} w_{ik}.x_{ij}.x_{kz}.D_{jz}^{s} \leq D_{ik}^{t}, \qquad \forall\ i, k \in [1, n] \\ -b, & \text{otherwise.} \end{cases} \tag{4.23}$$

$$R_3 = \begin{cases} +c, & \text{if } \sum_{j=1}^{m} x_{ij} = 1, \qquad \forall\ i \in [1, n] \\ -c, & \text{otherwise.} \end{cases} \tag{4.24}$$

Where a, b, and c are constant tuning parameters that are used to rank order the rewards in terms of importance. Whenever one of these constraints is violated, the agent receives a negative reward, and the training episode terminates. This is done to ensure the agent gives priority to realizing the SLAs over the rest of the rewards. Furthermore, terminating the episode helps in addressing the credit-assignment problem. In RL models, credit-assignment problem illustrates the difficulty in attributing rewards given by the environment to actions taken by the agent. It is caused by the delay between the behavior and the outcome during the training, i.e., the agent usually receives a reward at the end of the episode; thus, making it challenging to identify which action is responsible. However, through terminating the episode each time the agent violates a constraint, we can partially simplify the

environment for the agent to learn what went wrong and why a penalty was given; hence, addressing the credit-assignment problem.

If the action the agent took fulfilled the above constraints, the rest of the rewards are calculated. We define two more rewards; R_4 and R_5 corresponding to minimizing the latency and the adjustment to the network respectively. Firstly, we defined R_4 to minimize the overall latency in the network given all the constraints are realized. This is done since O-RAN puts a significant emphasis on the latency between the end user and the core network. Furthermore, reducing the latency decreases the link occupancy; thus, contributing to a lower OPEX. Therefore, according to Equation 4.25, the higher the latency, more penalty is given to the agent. Secondly, since we can have more than one placement that can serve a specific demand, we define R_5 to ensure the change to placement is minimal at each point in time; hence, the agent receives the maximum reward if the change in the network is equivalent to the scaling decision. Otherwise, the penalty is proportional to the change. We formulate the rewards as follows:

$$R_4 = -D, \quad \text{where } D = \sum_{z=1}^{m} \sum_{k=1}^{n} \sum_{j=1}^{m} \sum_{i=1}^{n} w_{ik}.x_{ij}.x_{kz}.D_{jz}^{s} \tag{4.25}$$

$$R_5 = \begin{cases} +d, & \text{if } \Delta VNF = \text{SD} \\ -\Delta VNF, & \text{otherwise} \end{cases} \tag{4.26}$$

Where d is a constant tuning parameter. We calculate the final reward at the end of each episode as shown in Equation 4.27. Moreover, the agent's neural network is updated after a number of episodes with the objective of maximizing the mean reward per episode.

$$R = R_1 + R_2 + R_3 + R_4 + R_5 \tag{4.27}$$

4.5 Summary

In this Chapter, we have further detailed the methodology presented in the previous Chapter. While discussing each stage, we have highlighted the challenges faced in designing different models along with approaches to overcome those challenges. Moreover, we have outlined the setup for each stage along with the rationale behind the choice in parameters and environment setup. In the next Chapter, we present the implementations of each stage, discussing different evaluation setup along with data preprocessing. Afterwards, we present the performance results of the system and its different stages followed with a discussion highlighting the efficacy of the system.

5 Performance Evaluation

In this Chapter, the models' performance is evaluated and compared. We start with exploring the environment in which the models are implemented. Then, the implementation of each stage is discussed. Finally, we present the results of the models. All ML models are implemented using Python, Tensorflow, and Keras. The adapted ILP model used for the data generation is implemented using C++, R, and Python. The optimization is done using IBM™ CPLEX through C++. The evaluation is performed on a computer with one Intel® Core™ i7-107000 CPU @2.90GHz, 16 GB RAM, and NIVIDIA GeForce RTX 2060 SUPER. We use different evaluation metrics to evaluate the performance of each model and/or stage. Firstly, at each stage, we evaluate the implemented ML models based on their predictive capabilities using metrics such as the mean absolute error. In the case of the RL agent, we use the mean reward per episode and the mean length per episode as evaluation metrics, in addition to maintaining the SLAs and minimizing the overall latency. Moreover, the models' generalization is then tested on a different dataset

to validate their performance. Secondly, we evaluate the performance of the system as a whole through the network utilization for each O-RAN VNF type, in addition to a comparison between the traffic and the number of deployed VNFs.

5.1 Implementation

5.1.1 Traffic Forecasting

5.1.1.1 Traffic Data: Analysis and Preprocessing

As previously mentioned, the quality of the data utilized for training and testing is crucial to the performance of the model. By analyzing the components of the series through decomposition, we can infer its predictability. Time-series can be decomposed into 3 main components: *i*) a trend component, representing the traffic growth over time, *ii*) a seasonal component, representing the cyclical behavior of the series, and *iii*) a remainder component, representing abnormalities due to sudden events (such as cell outage in the case of RANs). For a series to be predictable, it must be stationary, meaning the statistical properties of the series (mean, variance, and autocorrelation) are not a function of time. Moreover, the parameters for the models can be deduced from the statistical properties of the time-series.

There are two measures that show how suitable a time-series is for prediction: the autocorrelation function (ACF) and the Approximate Entropy of the time-

series. The ACF shows how correlated the series is with lagged versions of itself; the higher the correlation, the easier it is to predict. Approximate Entropy is used to quantify the regularity and predictability of fluctuations in a time-series; The higher the approximate entropy, the more difficult it is to forecast. We have examined three candidate datasets (Internet2 [42], Geant Network [8], Italia Telecom [44]) to determine their predictability and their applicability based on the mentioned measures. All three datasets contain traffic matrices representing the mobile traffic data between pairs of nodes for different timestamps. The internet2 has a temporal granularity of 5-minutes, while the Italia Telecom has 10-minuites, and the Geant has 15-minuites. The scatter plot shown in Figure 5.1 shows a comparison between the ACFs of the three dataset using lag values 1 to 4.

Expectedly, as the lag value increases, the time-series becomes less correlated. However, it is noticeable that the Italia dataset and the Geant network datasets are still correlated for higher values of the lag. This is also evident when decomposing the components of the internet2 dataset; the decomposition shown in Figure 5.2 illustrates the components of the series. As seen, although the seasonality is quite high, there is a significant residual component in the series, making it difficult to predict. The residual component is more evident when examining the Approximate Entropy of the dataset. Table 5.1 shows the Approximate Entropy for each dataset. Since the Italia Telecom dataset has the lowest Approximate Entropy and has a

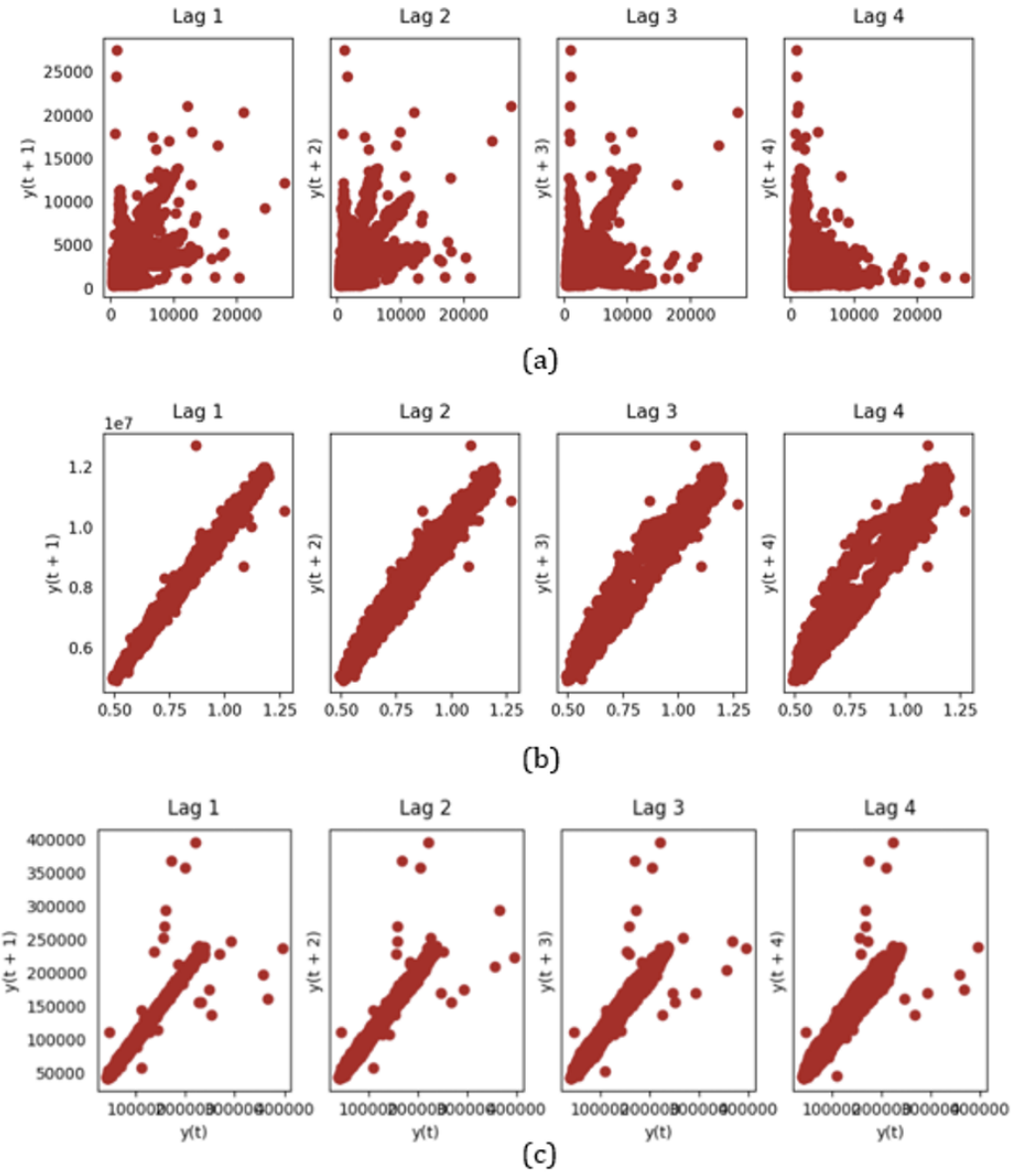

Figure 5.1: Lag plot for the autocorrelation of (a) internet2 dataset, (b) Geant dataset, and (c) Italia dataset.

higher temporal granularity compared to the Geant dataset, it proves to be the most suitable among the three.

Table 5.1: Approximate entropy for each dataset

	Intearnet2	**Geant**	**Italia**
Approximate Entropy	1.346	0.375	0.196

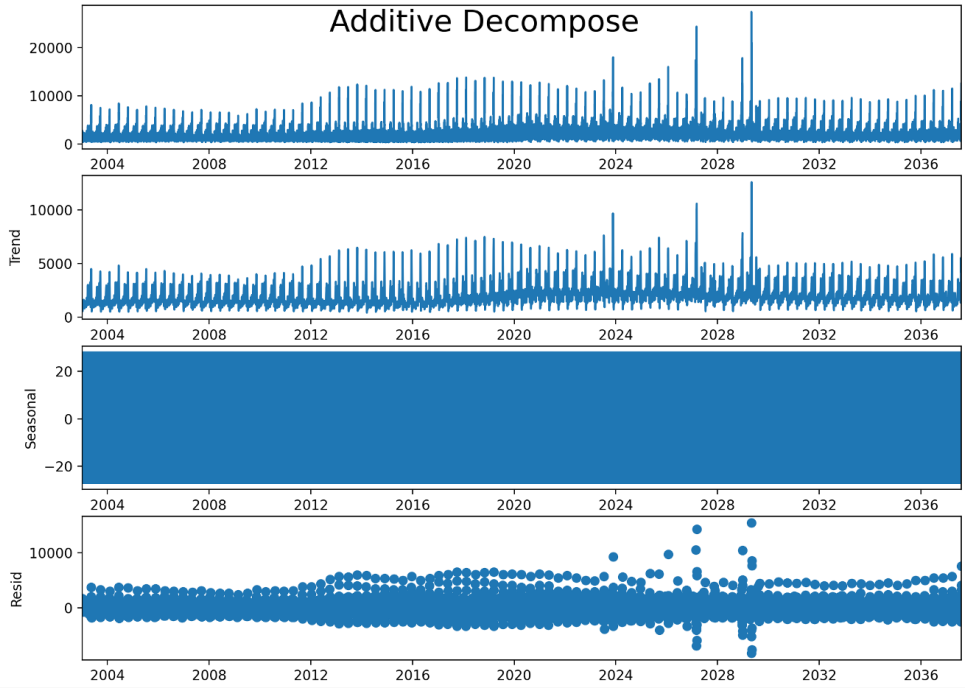

Figure 5.2: Additive decomposition of Internet2 traffic dataset.

The Italia dataset contains 2 sub datasets; Milano and Trento each contains a real-life call detail records collected for billing purposes in both provinces. Trento dataset includes 11466 cells while Milano has 10000 cells distributed to cover the

entire area. The data is collected in over 10 minutes intervals from 31st of October 2013 up until 1st of January 2014. Each timestamp contains the following features:

- SMS-in,
- SMS-out,
- Call-in,
- Call-out,
- Internet Traffic Activity.

Figure 5.3 shows a sample of the traffic through cell number 10000 in Trento. Table 5.2 shows a description of the traffic trace in Trento and Milano sub dataset.

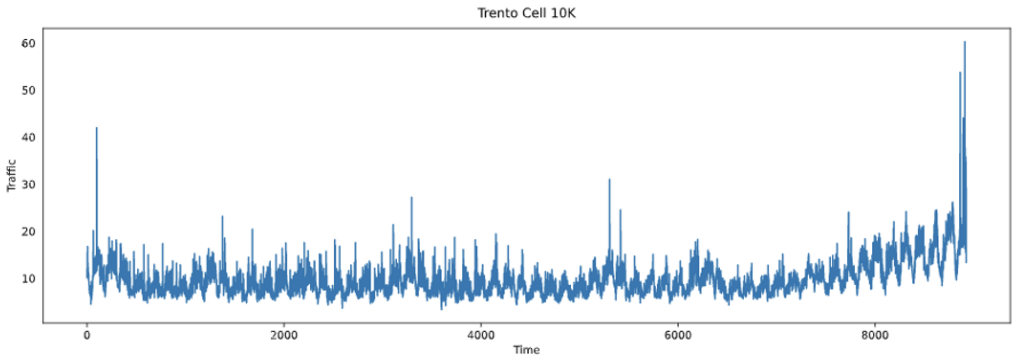

Figure 5.3: Internet traffic trace for cell 10000 in Trento trace.

Table 5.2: Statistical description for the aggregated data in Trento and Milan traces

	Trento	**Milan**
Count	8928	8928
Mean	95872.12	621964
Standard Deviation	37812.2	223385
Minimum	39934.18	20811
Lower than 25%	65270.7	413368
Lower than 50%	98718.65	64178
Lower than 75%	111219.2	810414
Maximum	395858.67	1234958

The traffic of a single cell is difficult to predict due to many external events that influence the fluctuations in that particular cell. These fluctuations cause the series to have lower autocorrelation; thus, decreasing the value of past knowledge in the prediction. To overcome this challenge, we aggregate the traffic spatially over the cells, reducing the influence of individual events happening in cells and increasing the autocorrelation through maximizing the cyclical patterns. The scatter plot shown in Figure 5.5 shows a comparison between the autocorrelations of

the individual cells vs. that of the aggregated series for lag 1 to 4. It is evident that the aggregated series is more correlated with itself for different lags; hence, it is easier to predict. Figure 5.4 shows the trace for the aggregated series. As expected, the aggregation has reduced the fluctuation and magnified the cyclical patterns. The aggregated series has an Approximate Entropy of 0.196, while the Approximate Entropy for cell 10000 is 1.386. Augmented Dickey Fuller (ADF) test is used to check for stationarity of the series. The ADF test shows the aggregated series to be stationary. Overall, the aggregated time-series of the Italia dataset shows promising results for the predictability of the time-series.

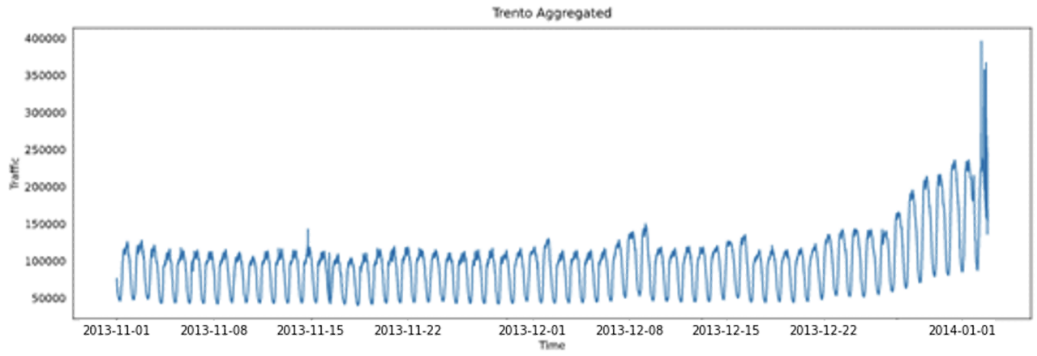

Figure 5.4: The aggregated traffic data of Trento trace.

5.1.1.2 ML Models

After ensuring the quality of the dataset and concluding that the aggregated Italia Telecom is the most suitable, the next step is utilizing the dataset to implement the

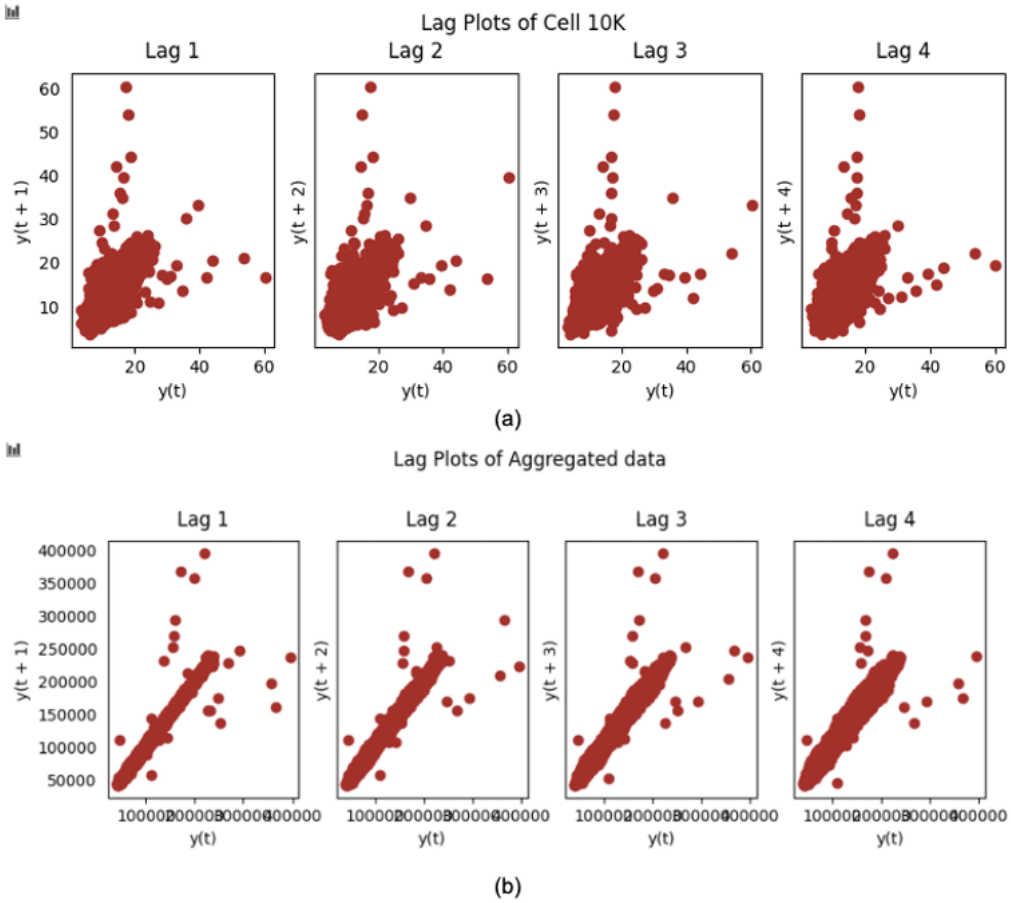

Figure 5.5: Lag plots for the autocorrelation of (a) cell 10000 and (b) aggregated data.

ML models. As mentioned, the proposed ML models are ARIMA and LSTM. In this Section, model specific transformations and data preprocessing are discussed, along with model parameters' extraction from the dataset.

ARIMA: In designing the ARIMA model, the objective is to determine the parameters of the function $ARIMA(p, d, q)$, where p is the order of auto-regression, d is the level of difference, and q is the order of the moving average. Firstly, the value of d dictates the number of times the series is differenced, which in turns ensures the stationarity of the series by eliminating the trend. We determine the value of d by applying the difference repeatedly till the series becomes stationary. The number of times the series is differenced before it becomes stationary represents the value of d. We check for stationarity after each differencing using the ADF test; where if the p-value is less than 0.05, the series is stationary. Secondly, the value of p can be determined through the Partial Autocorrelation Function (PACF). By plotting the PACF, the number of lags over the significance line dictates the value of p. Lastly, similar to p, we deduce the value of q from the plot of the ACF. Based on the ACF and PACF shown in Figure 5.6, we obtain the parameters for the model. The values of p, d, and q were chosen to be 6, 1, and 1 respectively.

Additionally, prior to inputting the series to the model, we applied some preprocessing to ensure the stationarity of the series and its compatibility with the

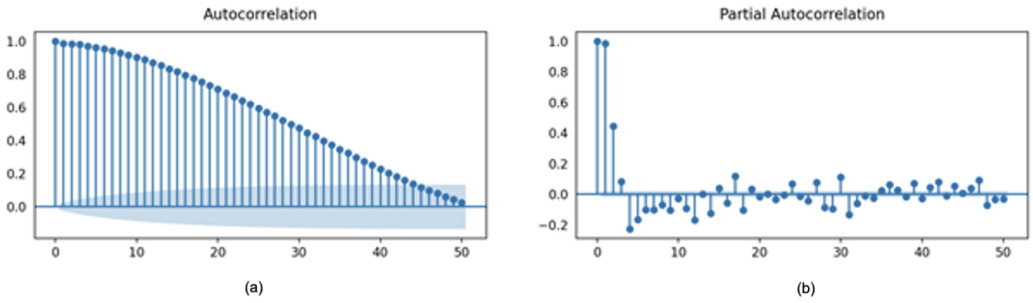

Figure 5.6: The autocorrelation and partial autocorrelation functions for the aggregated data.

model. As discussed previously, the trend is eliminated through differencing, however the seasonality is not. Therefore, we eliminate it beforehand. We eliminate the seasonality through a nonlinear transformation that guarantees the statistical properties of the series are not time dependent. Given a time-series V, firstly, we normalize the series as follows:

$$n_t = \frac{v_t - \mu}{\sigma} \tag{5.1}$$

Where μ is the mean and σ is the standard deviation. Then, to eliminate the trend component, we difference the normalized series n_t as follows:

$$d_t = n_t - n_t - 1 \tag{5.2}$$

Finally, we eliminate the seasonality through applying the following transformation to the differenced series d_t to ensure that mean, standard deviation, and

autocorrelation of the series are time independent:

$$d'_t = \frac{d_t}{\sigma_a} - \mu_h \tag{5.3}$$

Where σ_a and μ_h are the annual volatility and the hourly average respectively. Afterward, the resulting series d'_t, shown in Figure 5.7, is input to the model.

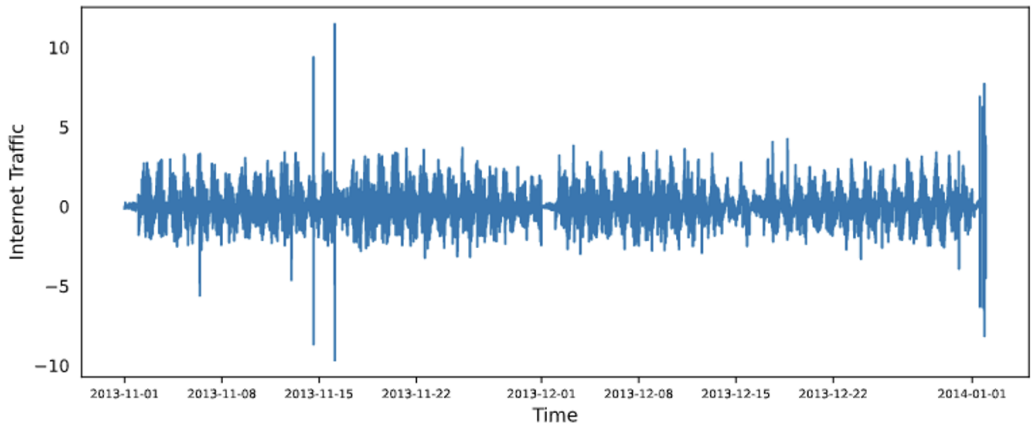

Figure 5.7: Transformed series.

LSTM: Prior to inputting the series to the model, we apply some preprocessing in terms of sliding window and feature extraction to reshape the dataset from a sequential dataset to a supervised learning dataset with the present and past N values as features and the value at $t + M$ as label. Since the value of the autocorrelation in the dataset gets lower as the lag increases; thus, reducing the accuracy of the prediction, only one timestamp prediction is chosen as the goal to ensure the accuracy. Therefore, we set the value of M to 1. As for N, the ACF shows that

the last somewhat significant correlation is at 11 lags; thus, signifying that adding more lags will not contribute to the accuracy of the prediction. On the contrary, it will just increase the model complexity and training time. Hence, we set the value of N to 11. Moreover, we include two more features to capture the periodicity in the series and the effects of seasonality on the prediction; Day_of_week to make a distinction between weekday versus weekend traffic, and $Hour\_of\_day$ to differentiate between daytime as opposed to nighttime traffic. We employ the Granger causality test to ensure the usability of those features in predicting the time-series. Finally, we rescale the dataset to fit the criteria of the neural network. Regarding the model parameters, the model has 2 layers containing 100 LSTM units in each layer with a learning rate of 0.0001 using the Adam optimizer and a ReLu activation function. We use Glorot Uniform initializer to initialize the weights instead of random initializers. The data is split into a 25% testing set and 75% training set and is fed to the model in batches of size 16. The loss function used is the mean absolute error.

5.1.2 Scaling Decision: Evaluation Setting and Data Generation

We implement the scaling decision model in 3 stages: generating SFC requests, generating training data, and training the ML model. Firstly, to generate SFC requests, we utilize the aggregated Trento traffic used for prediction. As discussed

in Section 4.3, we start by defining a peak arrival rate and average duration for the SFC requests. Afterwards, the requests are generated through sampling from a distribution with these parameters. The distribution defined for both interarrival rate and request duration is negative exponential [45]. The values and parameters for defining the distributions and generating the requests are shown in Table 5.3 [45]. After we generate the requests, we obtain a set of requests for each traffic instance. Then, we utilize the ILP to generate the optimal number and placement of VNF that can serve the requests given. For each traffic instance, given the network topology (the Internet2 network topology [42]), the ILP model takes the set of requests, and using CPLEX, produces an output file that contains information relevant to the optimal number, placement, and utilization of VNFs along with other statistics independent of the VNFs, such as the number of active requests as well as its departure and arrival time in addition to the bandwidth utilization. After running the ILP for all traffic instances, we then extract from all the output files the relevant information to construct a feature/label set to train the ML models.

Figure 5.8 shows a block diagram illustrating the workflow to generate the training dataset. Appendix B shows a sample of the output file obtained, along with a list of features extracted from all the output files. Each traffic instance has an execution time of 32s to generate an output file. To generate the training dataset from all the traffic instances, the model is run for all the traffic instances resulting

Table 5.3: Parameters for the SFC request generation

Parameter	**Value**
Distribution for request bandwidth	Uniform
Distribution of interarrival rate and request duration	Negative Exponential
Coefficient of variation for interarrival rate and request duration	0.25
Peak arrival rate	1 per 30 sec
Average request duration	1000 sec
Function chain	ODU-OCU-NearRIC
Occurrence probability	100%

in an execution time of approximately 5 days.

Since the forecasting is done at the traffic prediction stage, the relationship between the SFC requests and the number of VNF instances can somewhat be approximated as a linear relationship. Hence, a Multi-Level Perceptron (MLP) can be leveraged to model the relation. The implemented MLP takes the traffic requests as an input, and outputs the number of VNF instances required per type. The

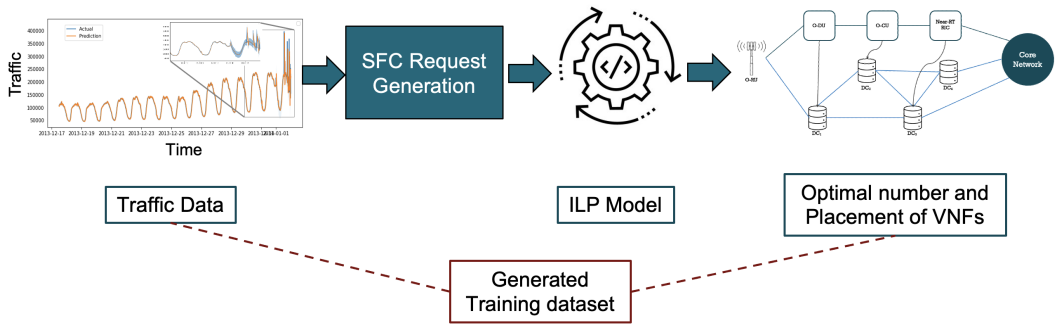

Figure 5.8: Block diagram for training data generation.

parameters for the MLP implementation are shown in Table 5.4. Since the scaling decision prediction is done per VNF type, we implement three models corresponding to each VNF type (O-DU, O-CU, Near RIC). The generated training dataset is split into a training and a testing set with a ratio of 80% / 20%.

5.1.3 RL Placement: Evaluation Setting

The VNF placement model is implemented using Python *Stable_Baselines*3 library. This library supports many different type of agent models based on the type of environment employed. In our case, since the environment is custom built based on the network topology and the physical network transformation discussed in Section 4.4, the agent model we used is the Proximal Policy Optimization (PPO). As mentioned previously, the RL approach chosen is Model-Free Policy-Based RL. PPO is an on-policy algorithm suitable for environments with multiple types of

Table 5.4: MLP model's parameters

Parameter	**Value**
Hidden layers	5
Number of nodes in each layer	200, 300, 200, 100, 32 nodes respectively
Activation function	ReLu
Learning rate	0.00001
Clip value	0.5
Optimizer	Adam Optimizer
Weights initializer	Xavier Initializer
Loss function.	Mean Absolute Error

observation and action spaces. PPO supports many types of policies that suits different environments. Such as Convolutional Neural Networks (CNN) policy for image processing, MLP policy for vector states, and Multi-Input policy for handling multiple inputs. PPO combines ideas from Advantage Actor Critic (A2C) (having multiple workers) and Trust Region Policy Optimization (TRPO) (it uses a trust region to improve the actor). The main idea is that after an update, the new policy should be not too far from the old policy. For that, PPO uses clipping to avoid too

large updates.

As for the environment, we set it up according to the discussion in Section 4.4. We use the internet2 network topology. We start by training the agent on a 15 pseudo-VNFs topology, then expand it to 70 pseudo-VNFs. The number of nodes the initial placement model places is 20 nodes.

Finally, for the reward function, we set the tuning parameters a, b, c, and d to reflect the importance of each reward function. The parameters a, b, and c represent the reward or penalty the agent gets from maintaining or violating the SLA; a, b, and c corresponds to capacity, latency, and redundancy constraints respectively. The parameter d corresponds to the reward the agent gets when the change caused by the action is equivalent to the scaling decision. To guarantee no SLA violation occurs; a, b, and c are much larger than d. Hence, the agent will always prioritize not violating the constraints to minimizing the change. Moreover, the constraints are ranked in terms of priority for the agent through sequentially placing them in the reward function; R_3 is prioritized followed by R_1, then R_2. The rationale being, we first need to ensure a placement happens, then check if it is feasible in terms of available resources, and finally ensure it does not violate the latency constraints. In accordance with this rationale, we set the values of the tuning parameters.

The model starts by initializing the environment; defining the topology and the nodes along with its placements acquired from the initial deployment model.

The agent then observes the environment (set of active pseudo-VNFs) along with the scaling decision and active requests to generate an action (a new set of active pseudo-VNFs). The action then triggers a step function. The step function adjusts the environment according to the action provided by the agent, then computes the corresponding reward. If any of the constraints are violated, the training episode is then terminated, and the agent is given a negative reward for the episode. Afterwards, the agent updates the PPO weights every 2048 timestep. The cycle continues until the agent becomes able to place all VNF instances without violating SLA and can accumulates the highest reward (maximizing the reward).

5.2 Results

Figure 5.9 shows the results of training and testing for LSTM and Figure 5.10 shows results for ARIMA. LSTM has training and testing error of 2489 and 2491. As for ARIMA, testing error is 3409 with a relatively narrow prediction confidence. Both models have given a relatively low error in comparison with the mean (95872) and the standard deviation (37812) of the Trento trace.

Given Figure 5.9 & Figure 5.10 and the mentioned results, both models have achieved a similar performance; however, LSTM has a slightly lower error compared to ARIMA. However, ARIMA has better modeled the sudden spikes in the trace. This is expected as the LSTM captures the general patterns during the training,

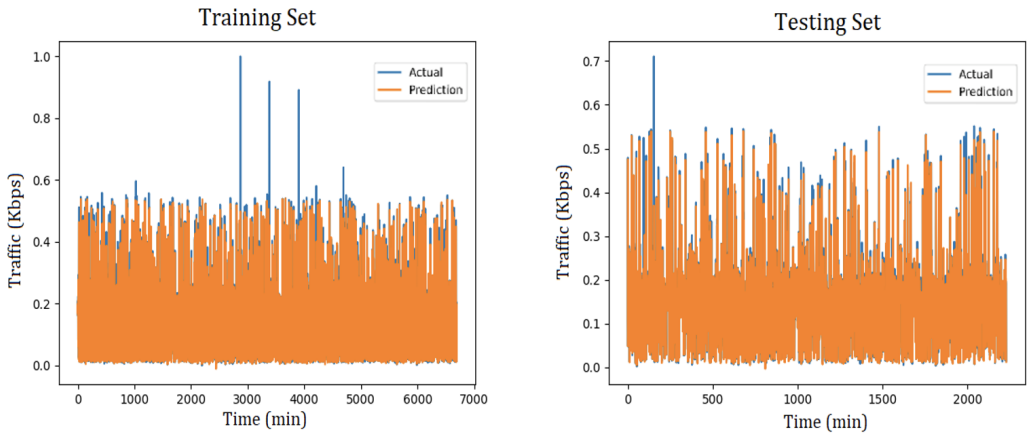

Figure 5.9: Predictions vs. Actual for training and testing sets (LSTM).

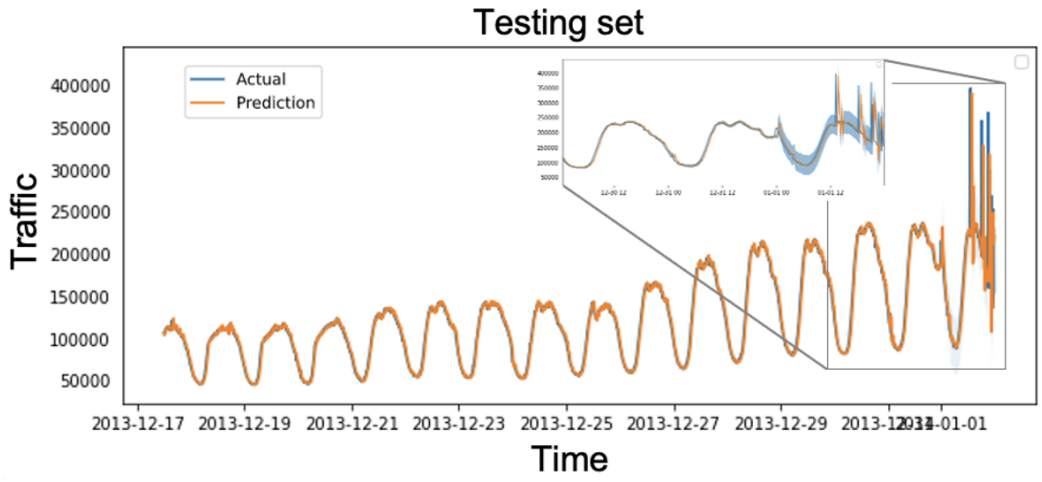

Figure 5.10: Predictions vs. Actual for testing set (ARIMA).

while ARIMA builds the model at each timestep. Therefore, it can gradually learn such anomalies. Although ARIMA captures the spikes better, the prediction confidence region grows slightly wider whenever there is a sudden spike (Figure 5.10).

As for execution time, ARIMA requires 1360s for the whole set and LSTM took 375s for training and 3.9s for testing. This discrepancy is again due to ARIMA building the model at each timestep. Moreover, to test the generalization of the trained LSTM, it is tested on the Milan dataset after being trained on the Trento. Figure 5.11 shows the testing results. The error is found to be 22786, relatively low compared to the mean (621964), implying the model is generalizing well.

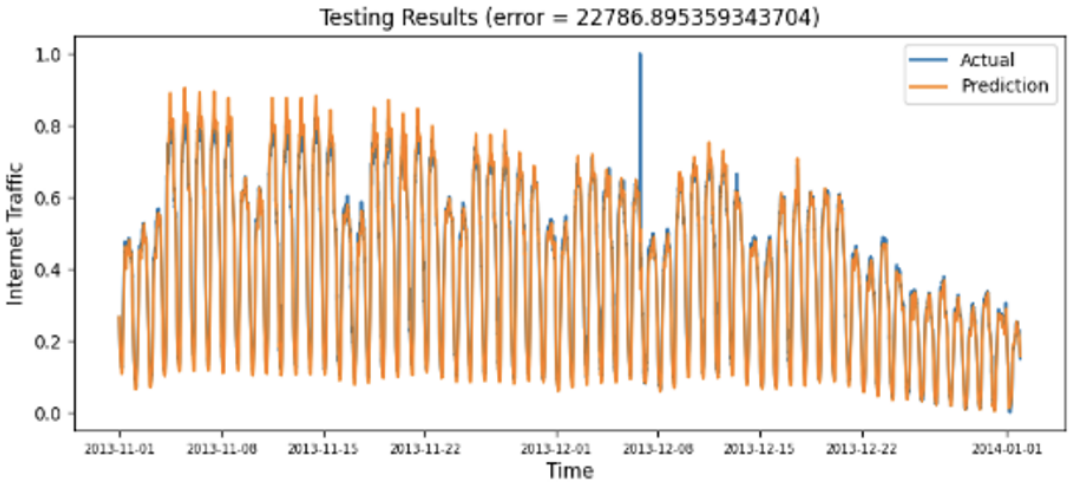

Figure 5.11: Predictions vs. Actual for the Milan dataset (LSTM).

Table 5.5: Training and testing errors per VNF Type

VNF Type	**O-DU**	**O-CU**	**Near RIC**
Training Error	0.1881	0.0789	0.0962
Testing Error	0.1884	0.0853	0.0996

Figures 5.12, 5.13, and 5.14 shows the prediction vs. the actual number of VNF instances for O-DUs, O-CUs, and Near RICs respectively. Each Figure shows both the training and testing set for the scaling models per VNF type. Table 5.5 shows the training and testing error for each model. Given the training and testing errors, the model fairly captures the pattern when predicting the number of required VNF instances per type. The average error is around $\pm$ 0.1 VNF and the traffic closely matches the number of deployed VNFs at each point in time (Figure 5.15).

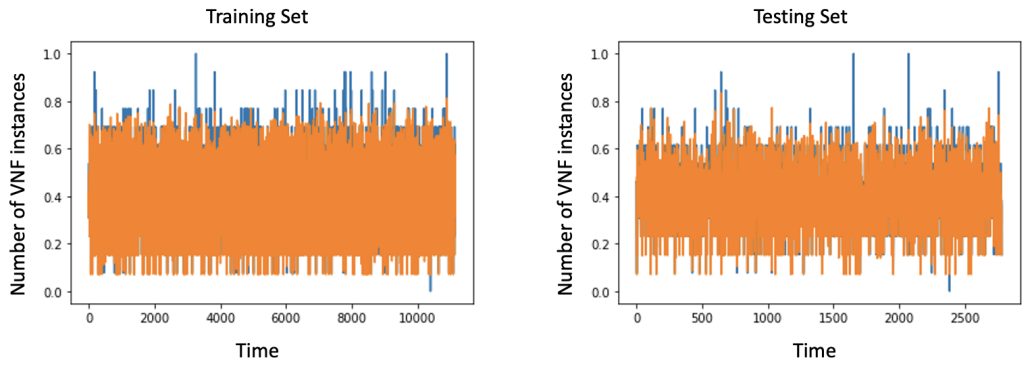

Figure 5.12: Predictions vs. Actual for training and testing set (O-DU).

Figure 5.15 displays a comparison between the total number of active VNF and the traffic. As shown, both display a similar pattern of behavior implying that the resource utilization is maximized and the number of instances deployed matches the traffic. Figure 5.16 shows the resource utilization in the network with respect to the requested bandwidth. We define the utilization as the ratio between the requested bandwidth and the total capacity per VNF type. As seen, the utilization is quite

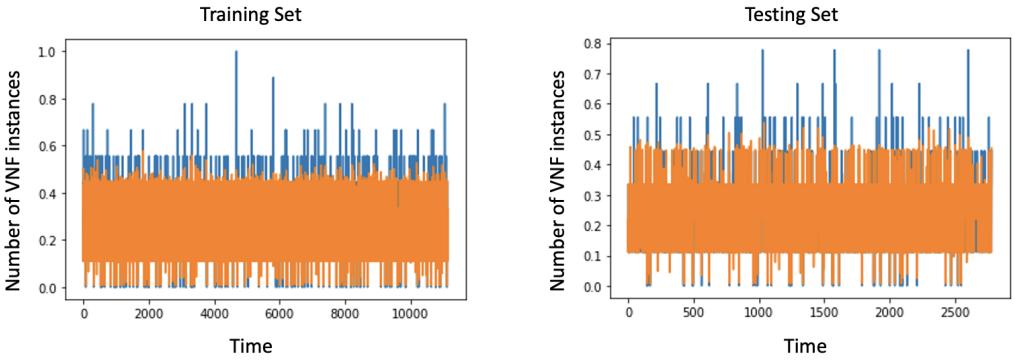

Figure 5.13: Predictions vs. Actual for training and testing set (O-CU).

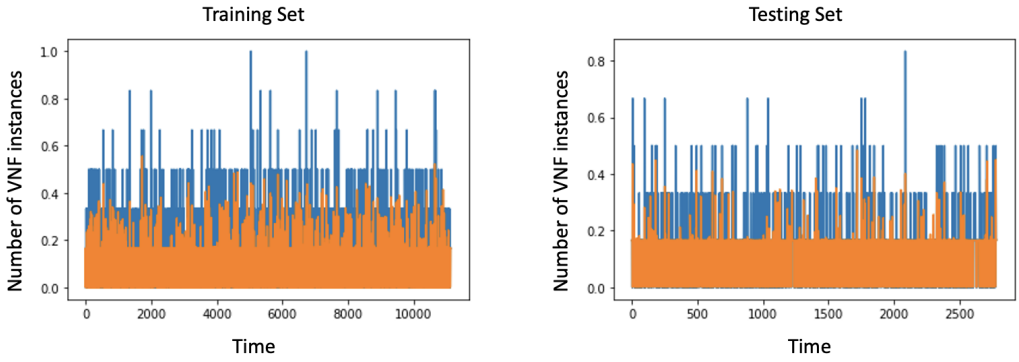

Figure 5.14: Predictions vs. Actual for training and testing set (Near RIC).

high for all VNF types except when the traffic drops. The increase in utilization then translates into reduction in the cost in terms of OPEX.

Figure 5.17 shows the performance of the model in the training phase with 15 pseudo-VNFs. As shown, the model learned quickly in the span of 100K step. The graph shows the mean reward per episode. Around the 60K step, the model learned all the constraints and maximized the reward function. Figure 5.18 shows

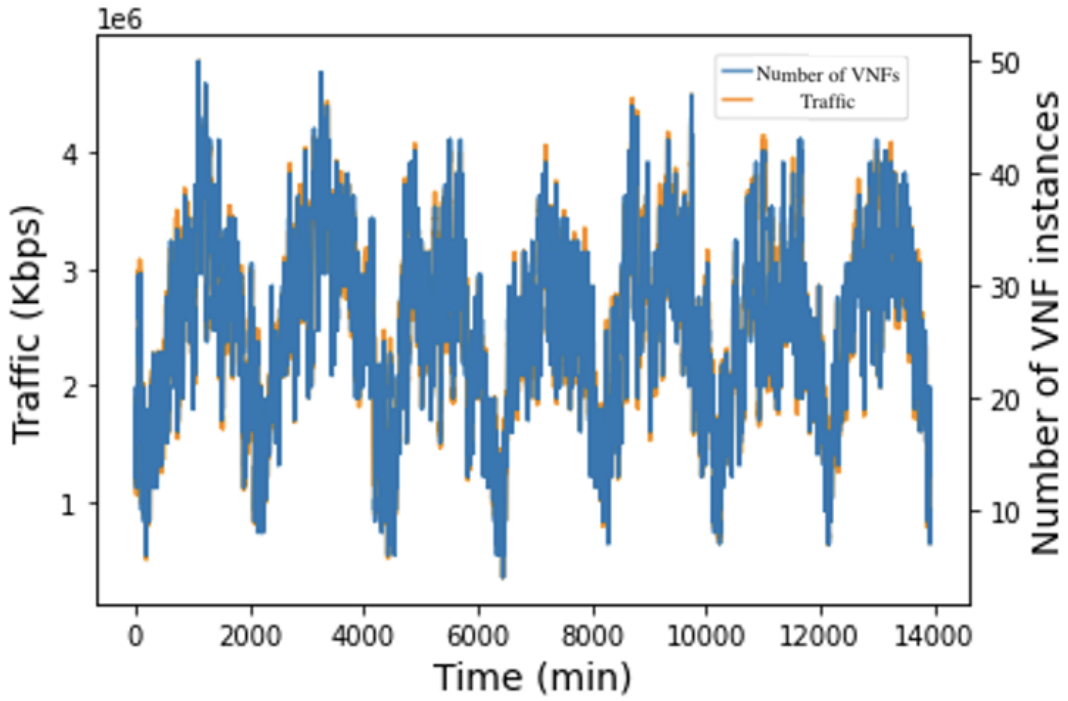

Figure 5.15: Comparison between deployed VNF instances and aggregated traffic.

a comparison between the reward accumulated by the agent in the first episode and episode 100.

Afterward, we expand the model to support more VNFs. The extended model supports 70 pseudo-VNFs. Figure 5.19 presents the agent's performance placing the 70 pseudo-VNFs. The figure shows the mean reward per episode for 500K steps. As seen in the figure, there is a slight increase in the reward attained by the agent over the span of the 500K steps. However, the training is too slow, and the reward is too fluctuant.

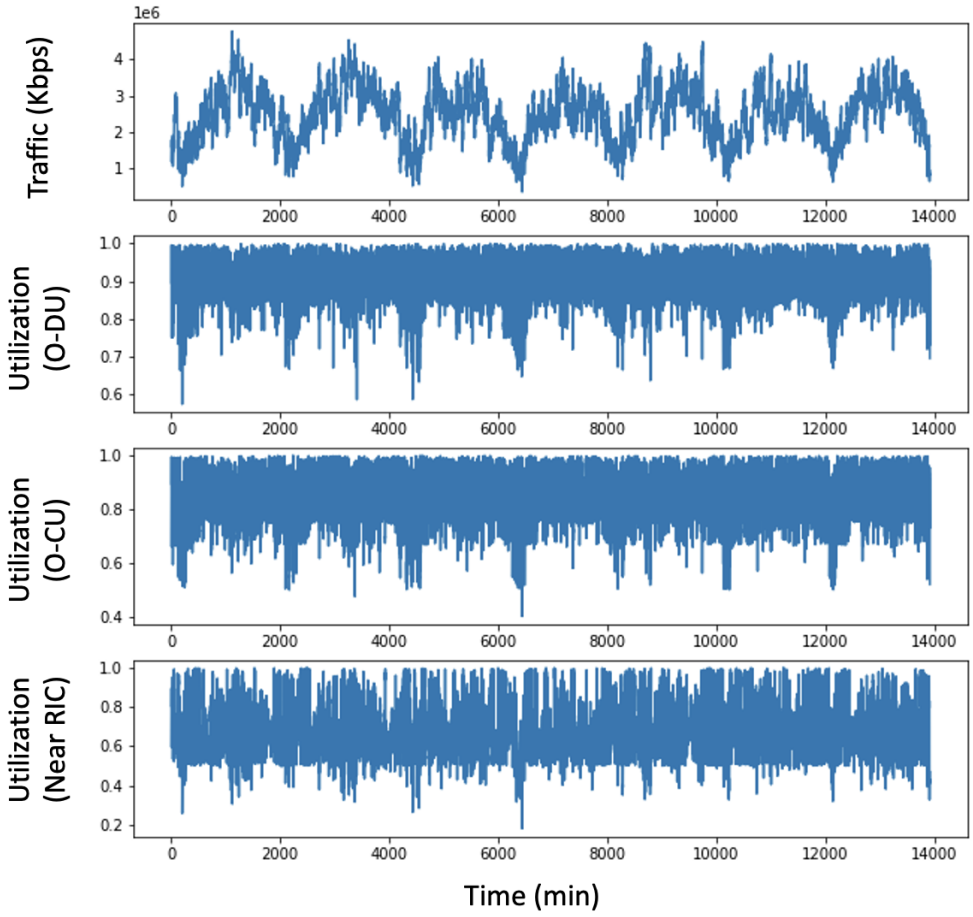

Figure 5.16: Network utilization.

The long training time of the RL model can be explained by the credit assignment problem. To address the credit assignment issue, we use a variable episode length rather than a fixed one. As discussed previously, terminating the episode whenever a constraint is violated helps in addressing the credit-assignment problem. Using the fixed episode length, the agent receives the reward at the end of the episode; thus, making it challenging to identify which action was responsible.

Figure 5.17: Mean reward per episode for 15 pseudo-VNFs.

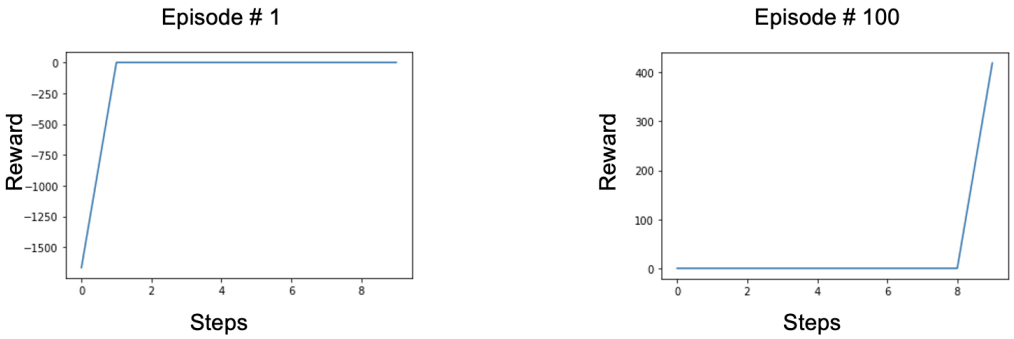

Figure 5.18: Reward for two sample episodes; 1 & 100.

However, through terminating the episode each time the agent violates a constraint (variable episode length), we can partially simplify the environment for the agent to learn what went wrong and why a penalty was given; hence, addressing the credit-assignment problem. Figure 5.20 presents the agent's performance placing 70 pseudo-VNFs using variable episode length. It shows the mean reward per

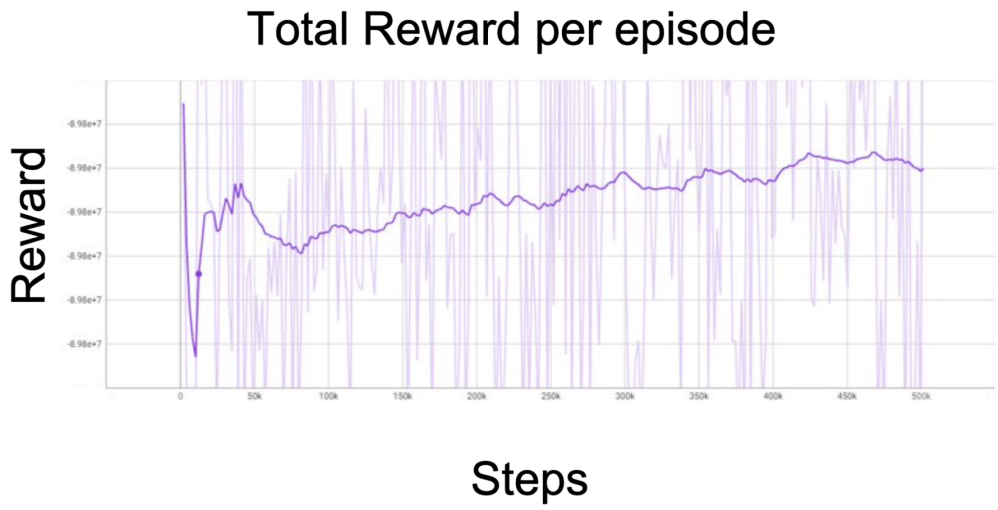

Figure 5.19: Mean reward per episode for 70 pseudo-VNFs.

episode and the mean length per episode for the training. The agent is trained for 3.5 million steps. The model maximizes the reward around 1.5 million steps. Furthermore, the more complex the environment get, the more the agent tends to exploit rather than explore. Hence, to reduce the complexity of the environment the node selection is fed to the agent rather than the agent determining it.

The variable length for the episode splits the training into three stages as shown in Figure 5.21. Firstly, in region A the agent is learning to place the correct number of VNF instances. Around 1 million steps the agent learns the correct number of VNFs to be placed, however, it is violating capacity and latency constraints. Afterward, the episode length starts to increase, and the agent starts learning the constraints as shown in region B. Once the agent stopped violating all the

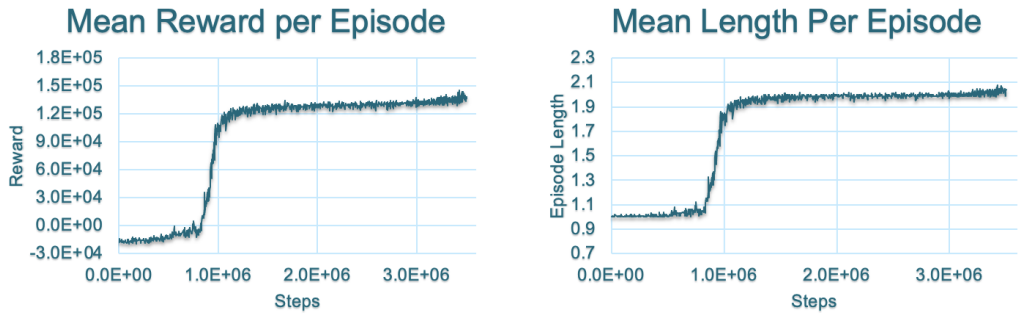

Figure 5.20: Agent's performance for 70 pseudo-VNFs with variable episode length.

constraints (1 million step), it starts learning how to minimize the latency (region C), i.e., maximizing the reward. Once the agent learns to maximize the reward and the episode length saturates. Hence, it signifies that the agent now is capable of placing instances without violating any constraints while minimizing latency and changes to the network.

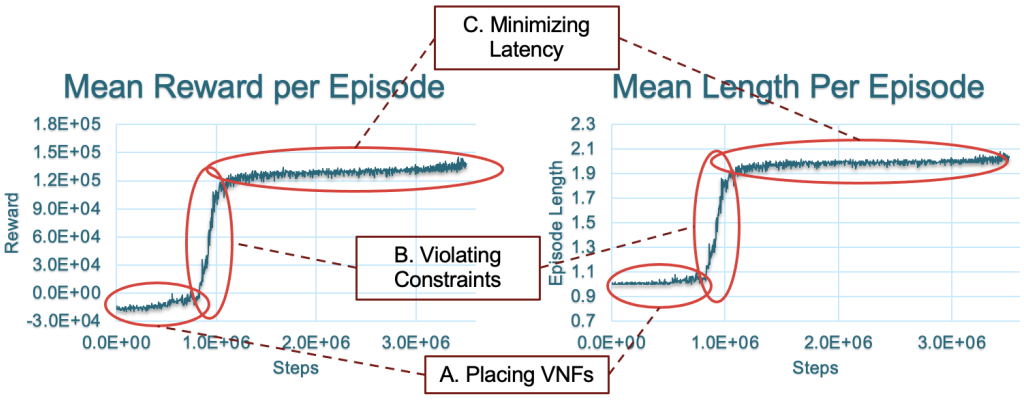

Figure 5.21: Mean reward per episode interpretation for 70 pseudo-VNFs.

5.3 Summary

In this Chapter, we have presented the implementations of each stage, discussing different evaluation setups along with data preprocessing. Moreover, the choice of parameters for each model and its environment is discussed. Afterwards, the models' performance is evaluated and compared. We start with exploring the environment in which the models are implemented. Then, the implementation of each stage is discussed. Finally, we present the results of the models. Furthermore, we have evaluated the performance of the system and its different stages followed with a discussion highlighting the efficacy of the system in context of O-RAN. The evaluation carried out in this Chapter uses the aforementioned Italia Telecom dataset and is agnostic to any specific application. However, the framework followed applies to application specific data and can be carried out in specific scenarios and use cases. Use cases such as Virtual Reality, Internet of Things, or Augmented Reality can be examined through using this framework, especially since the *O-RAN Alliance* focused on demanding applications such as these when defining the O-RAN specification.

Overall, the system has achieved the desired results and realized our objectives. Ultimately, our objective is to design, implement, and evaluate a VNF elastic orchestration policy for O-RAN VNFs that is capable of proactively and dynamically

scaling resources depending on the incoming demand. The results we have portrayed in this Chapter have proved the efficacy of our elastic system, which has been demonstrated throughout the evaluation, particularly through the network utilization, and the comparison between the deployed VNFs and the traffic demand (Figures 5.15 & 5.16). Although the system performed well, there still is a room for improvement. However, the framework we have portrayed throughout this work provides a solid standing ground for further improvements in terms of implementing an end-to-end solution for 5G O-RAN. In the next Chapter, we conclude our work and provide a discussion of possible improvements and tracks for future work to extend the system's performance.

6 Conclusion and Future Work

6.1 Conclusion

The ever-increasing demand in mobile networks has put a significant strain on the current mobile network infrastructure. The upcoming 5G networks entails a deep restructuring of the network architecture, especially its RAN architecture. O-RAN has been proposed as a way of restructuring the current RAN while addressing many of its limitations and complying with the stringent 5G requirements.

Employing intelligent solutions for dynamic resource management is a key advantage of the O-RAN virtualization. Efficient and proactive allocation of resources is essential to ensure preserving the O-RAN requirements. As a step towards ensuring an elastic O-RAN, we proposed a proactive elastic orchestration framework for resource management in the context of O-RAN. Motivated by this, we have provided an overview of the O-RAN architecture and its components, with the focus of its challenges and the research opportunities associated with those challenges. Moreover, a discussion regarding employing various state-of-the-art technologies in

the O-RAN network is presented, highlighting its benefits and some related applications and services. Furthermore, we discussed several areas in which researchers employed ML models in solving O-RAN related challenges. Dynamic resource management is one of the rising use cases when it comes to exploiting ML. The proposed elastic model consists of traffic forecasting, scaling decision, initial placement, and dynamic placement of instances.

Traffic forecasting is a crucial element for the success of the elastic model. The prediction accuracy is pivotal in implementing elastic techniques aimed toward OPEX reduction. We investigated traffic forecasting by developing two models (LSTM and ARIMA) using Python. The models profiled and predicted the traffic based on past traffic. Moreover, we explored time-series forecasting problem, in addition to discussing different evaluation metric for the traffic dataset. Data preprocessing along with model specific transformation were explored. Using the time-series analysis, we ensured the quality and predictability of the dataset. The models were trained and tested using the traffic dataset provided in [44]. The models' performances were then compared. Results have shown that LSTM has slightly outperformed ARIMA in terms of overall error, while ARIMA better modeled the spikes. According to the training nature of the models and their execution time, it can be deduced that LSTM is better fit for this application as it has significantly lower execution time and the training is done offline. This is largely due to the fact

that ARIMA is statistical modeling tool, i.e., with each prediction the model has to be rebuilt. The obtained results are promising and can form a strong foundation in implementing an end-to-end proactive elastic orchestrator.

With a solid prediction, we investigated the initial placement of O-RAN VNFs. Given a topology and an underlying physical infrastructure, the optimization model placed the VNF instances realizing the various constraints. The ILP optimization model was implemented using IBM CPLEX. Since latency is one of the pivotal points when it comes to O-RAN, the model was designed to minimize the overall latency between the deployed VNFs, subjected to the constraints discussed.

Using the deployed VNFs and the traffic prediction, a scaling decision can be reached. However, the first step is to create traffic requests to propagate throughout the deployed VNFs. The traffic data realistically portrays the variation in volume over time. However, the variation in terms of traffic request arrival and duration as well as how many requests are active has to be synthesized. We defined the distributions and sampled the requests to generate the new set of traffic instances that can more accurately portray realistic network scenarios. Then using these requests, a training dataset was generated by using an ILP model on all the traffic instances and acquiring the optimal number of VNF per type for each traffic instance. Therefore, by utilizing this training set, we trained three ML model for each type of VNF to predict the optimal number of instances given a traffic instance. Comparing

the output of these model with the actual number of instances deployed a scaling decision can be deduced.

Finally, we developed an RL model based on the initial placement model that can place instances according to the scaling decision and the active requests while realizing the various constraints. Moreover, the agent was trained to minimize the overall latency and the adjustments to the network. The model was implemented using Python *Stable_Baselines*3 library. The environment used was a custom environment based on the network topology, and the reward function is based on the constraints.

Combining all the aforementioned stages, a proactive elastic orchestration model for dynamic resource management is realized. Acquiring the traffic forecast at the first stage maintained the proactivity of the model while simplifying the later stages. The results, illustrated in Chapter 5, shows the promising nature of elastic models in terms of maximizing the resource utilization. Moreover, since the agent minimizes the latency when placing instances, it results in lower bandwidth utilization of the links throughout the network; thus, a further reduction to the OPEX. Hence, aligning with the objective of the models.

6.2 Future Work

Despite the promising nature of the model's performance, further improvement can be investigated to extend the performance. Firstly, in the traffic forecasting stage, future work can focus on improving accuracy and extending the model to account for drastic changes in the traffic trend (new cell site, equipment failure, social events, etc.). Furthermore, to acquire a higher resolution prediction, joint prediction can be employed to predict neighboring cells' traffic. It provides a good balance between coping with the noise on the cell level and a better prediction resolution. Additionally, extending the model's prediction horizon can prove effective even though it reduces the prediction accuracy.

Secondly, placement models can be extended to be topology independent, however, that can prove challenging when training the RL agent. Moreover, the requests generation stage used synthetic data to obtain the inter-arrival and duration for the requests. This is due to 5G O-RAN being a new and growing research topic; there is still a lack of information, data, and specification related to its units and use cases. As the field matures, obtaining such data for 5G O-RAN becomes easier, then, applying real-world data instead of synthetic one can allow the models to pick up on more patterns which will increase the accuracy and better the overall performance.

Finally, the RL model can now support only 70 pseudo-VNFs. Although it is somewhat representative of the network behavior, a higher number can further demonstrate the OPEX reduction that can be achieved. Since pseudo-VNF architecture is used, the RL is topology specific. This has the advantage of reducing the training time for the RL agent, however, it falls short for a generalized solution. The complexity of the model can be further reduced by deploying one instance at a time instead of the agent coming up with all the placements at once. Although, the utility of this approach is self-evident, it requires the RL to be Model-Based, or at least Value-Based. In this case, the agent has to keep future VNF (state value) in mind when placing the first ones. To achieve such a performance, a fairly accurate model of the environment has to be developed. Such model is quite challenging due to the dynamic nature of the network and the active requests.

Research effort in the development of these elastic models in the 5G context is still in its infancy. The virtualization of O-RAN has opened up all kind of possibilities to address various challenges. This work serves as proof to the utility in applying ML solutions to address some of these challenges. That being said, there is certainly a room for improvement.

Bibliography

[1] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.

[2] Y. Gu, Y. Hu, Y. Ding, J. Lu, and J. Xie, "Elastic virtual network function orchestration policy based on workload prediction," *IEEE Access*, vol. 7, pp. 96 868–96 878, 2019.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[4] S. Redana, Ö. Bulakci, A. Zafeiropoulos, A. Gavras, A. Tzanakaki, A. Albanese, A. Kousaridas, A. Weit, B. Sayadi, B. T. Jou *et al.*, "5g ppp architecture working group: View on 5g architecture," 2019.

[5] R. Y. Ameen and A. Y. Hamo, "Survey of server virtualization," *arXiv preprint arXiv:1304.3557*, 2013.

[6] C. Li and A. Akman, "O-ran use cases and deployment scenarios. towards open and smart ran," *White paper*, 2020.

[7] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. IEEE, 2015, pp. 255–260.

[8] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2018.

[9] M. Peng, Y. Li, Z. Zhao, and C. Wang, "System architecture and key technologies for 5g heterogeneous cloud radio access networks," *IEEE network*, vol. 29, no. 2, pp. 6–14, 2015.

[10] S. El Hassani, A. Haidine, and H. Jebbar, "Road to 5g: Key enabling technologies." *J. Commun.*, vol. 14, no. 11, pp. 1034–1048, 2019.

[11] V. Q. Rodriguez, F. Guillemin, A. Ferrieux, and L. Thomas, "Cloud-ran functional split for an efficient fronthaul network," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2020, pp. 245–250.

[12] M. Peng, K. Zhang, J. Jiang, J. Wang, and W. Wang, "Energy-efficient resource assignment and power allocation in heterogeneous cloud radio access networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 11, pp. 5275–5287, 2014.

[13] Y. Shi, Y. E. Sagduyu, and T. Erpek, "Reinforcement learning for dynamic resource optimization in 5g radio access network slicing," in *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2020, pp. 1–6.

[14] T. Li, X. Zhu, and X. Liu, "An end-to-end network slicing algorithm based on deep q-learning for 5g network," *IEEE Access*, vol. 8, pp. 122 229–122 240, 2020.

[15] H. Zhou, M. Elsayed, and M. Erol-Kantarci, "Ran resource slicing in 5g using multi-agent correlated q-learning," in *2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 2021, pp. 1179–1184.

[16] Q. Yuan, H. Tang, Y. Zhao, and X. Wang, "An approach for virtual network function deployment based on pooling in vepc," *IEICE Transactions on Communications*, 2017.

[17] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau, "Online vnf scaling in datacenters," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 140–147.

[18] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "End-to-end performance-based autonomous vnf placement with adopted reinforcement learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 534–547, 2020.

[19] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1346–1354.

[20] C. H. T. Arteaga, F. Risso, and O. M. C. Rendon, "An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an nfv-based epc," in *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE, 2017, pp. 1–7.

[21] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 486–494.

[22] X. Zhang, C. Wu, Z. Li, and F. C. Lau, "Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[23] A. Bilal, T. Tarik, A. Vajda, and B. Miloud, "Dynamic cloud resource scheduling in virtualized 5g mobile systems," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[24] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *2014 IEEE network operations and management symposium (NOMS)*. IEEE, 2014, pp. 1–9.

[25] J. Varia, S. Mathew *et al.*, "Overview of amazon web services," *Amazon Web Services*, vol. 105, 2014.

[26] E. Hormozi, H. Hormozi, M. K. Akbari, and M. S. Javan, "Using of machine learning into cloud environment (a survey): managing and scheduling of resources in cloud systems," in *2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE, 2012, pp. 363–368.

[27] G. Brataas, E. Stav, S. Lehrig, S. Becker, G. Kopčak, and D. Huljenic, "Cloudscale: scalability management for cloud systems," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013, pp. 335–338.

[28] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *2010 International Conference on Network and Service Management*. Ieee, 2010, pp. 9–16.

[29] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth, "A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, 2013, pp. 1–10.

[30] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow world congress*, vol. 48. sn, 2012, pp. 1–16.

[31] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 106–120, 2017.

[32] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[33] 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, 2019.

[34] D. M. Gutierrez-Estevez, M. Gramaglia, A. De Domenico, N. Di Pietro, S. Khatibi, K. Shah, D. Tsolkas, P. Arnold, and P. Serrano, "The path towards resource elasticity for 5g network architecture," in *2018 IEEE wireless communications and networking conference workshops (WCNCW)*. IEEE, 2018, pp. 214–219.

[35] M. A. Sharkh, Y. Xu, and E. Leyder, "Cloudmach: Cloud computing application performance improvement through machine learning," in *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 2020, pp. 1–6.

[36] M. Series, "Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond," *Recommendation ITU*, vol. 2083, p. 0, 2015.

[37] R. J. Hyndman, A. V. Kostenko *et al.*, "Minimum sample size requirements for seasonal forecasting models," *foresight*, vol. 6, no. Spring, pp. 12–15, 2007.

[38] O. Alliance, "O-ran use cases and deployment scenarios," *White Paper, Feb*, 2020.

[39] C. Gijón, M. Toril, S. Luna-Ramírez, M. L. Marí-Altozano, and J. M. Ruiz-Avilés, "Long-term data traffic forecasting for network dimensioning in lte with short time series," *Electronics*, vol. 10, no. 10, p. 1151, 2021.

[40] O. Alliance, "O-ran fronthaul working group; control, user and synchronization plane specification," *ORAN-WG4 CUS. 0-v01. 00, Technical Specification*, pp. 1–189, 2019.

[41] ——, "Operator defined next generation ran architecture and interfaces," *URL: https://www.o-ran.org*, 2020.

[42] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.

[43] H. Yu, F. Musumeci, J. Zhang, M. Tornatore, L. Bai, and Y. Ji, "Dynamic 5g ran slice adjustment and migration based on traffic prediction in wdm metro-aggregation networks," *Journal of Optical Communications and Networking*, vol. 12, no. 12, pp. 403–413, 2020.

[44] G. Barlacchi, M. De Nadai, R. Larcher, A. Casella, C. Chitic, G. Torrisi, F. Antonelli, A. Vespignani, A. Pentland, and B. Lepri, "A multi-source dataset of urban life in the city of milan and the province of trentino," *Scientific data*, vol. 2, no. 1, pp. 1–15, 2015.

[45] S. Lange, H.-G. Kim, S.-Y. Jeong, H. Choi, J.-H. Yoo, and J. W.-K. Hong, "Predicting vnf deployment decisions under dynamically changing network conditions," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–9.

A Initial Placement Optimization Model

A.1 Model

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: Khalid
 * Creation Date: Sep. 10, 2021 at 7:11:39 a.m.
 *********************************************/
{string} PM = ...;
{string} VNF = ...;
{string} CVNF = ...;
{string} RVNF = ...;
{string} PMr = ...;
{string} R = ...;
float Ds[PM,PM] = ...;
int u[R,PM] = ...;
```

```
int c[R,VNF] = ...;

int cl[PM] = ...;

int e[PM] = ...;

int w[VNF,VNF] = ...;

int m = ...;

int n = ...;

int Dv = ...;

dvar boolean X[VNF,PM];


minimize

   sum( z in PM ) ( sum( k in VNF ) ( sum( j in PM ) (sum( i in VNF ) (

      w[i][k] * X[i][j] * X[k][z] * Ds[j][z] ) ) ) );


subject to {

  forall( i,k in VNF )

   {

    sum(z in PM)

      sum(j in PM)

        (w[i][k] * X[i][j] * X[k][z] * Ds[j][z]) <= Dv;

    }

   forall( i in VNF )
```

```
  {
    sum( j in PM ) (X[i][j]) == 1;
  }
 forall(j in PM )
  {
    forall( r in R )
      {
        sum( i in VNF ) ( X[i][j] * c[r][i]) <= u[r][j];
      }
   }
  forall( i in CVNF)
  {
   sum( j in PM ) ( cl[j] * X[i][j] ) == 0;
  }
  forall( i in RVNF)
  {
   sum( j in PM ) ( e[j] * X[i][j] ) == 0;
  }
}
```

A.2 Data

```
/*********************************************
 * OPL 20.1.0.0 Data
 * Author: Khalid
 * Creation Date: Sep. 10, 2021 at 7:11:39 a.m.
 *********************************************/
PM = {"pm1","pm2","pm3","pm4","pm5"};
VNF = {"ric1","cu1","cu2","cu3","du1","du2","du3","du4","du5","du6"};
CVNF = {"ric1"};
RVNF = {"cu1","cu2","cu3","du1","du2","du3","du4","du5","du6"};
PMr = {"pm1","pm2","pm3"};
R = {"CPU","RAM","HDD"};


Dv = 10;
Ds = [[0,4,10,12,2], [4,0,1000,5,6], [10,1000,0,2,4], [12,5,2,0,2],
    [2,6,4,2,0]];
u = [[64,32,64,128,32], [512,128,512,1024,128],
    [2048,1024,2048,4096,1024]];
c = [[32,16,16,16,16,16,16,16,16,16], [128,64,64,64,64,64,64,64,64,64],
    [1024,512,512,512,512,512,512,512,512,512]];
```

```
cl = [1,1,1,0,0];

e = [0,0,0,1,1];

w = [[0,1,1,1,0,0,0,0,0,0], [1,0,0,0,1,1,0,0,0,0],

    [1,0,0,0,0,0,1,1,0,0], [1,0,0,0,0,0,0,0,1,1], [0,1,0,0,0,0,0,0,0,0],

    [0,1,0,0,0,0,0,0,0,0], [0,0,1,0,0,0,0,0,0,0], [0,0,1,0,0,0,0,0,0,0],

    [0,0,0,1,0,0,0,0,0,0], [0,0,0,1,0,0,0,0,0,0]];

m = 5;

n = 10;
```

B ILP Model Data Generation

B.1 SFC Requests Generation

Sample of the generated SFC requests tuples:

```
0,6,10,92677,705,0.00000010,ODU,OCU,NearRIC
0,0,2,73119,710,0.00000010,ODU,OCU,NearRIC
0,7,9,98379,735,0.00000010,ODU,OCU,NearRIC
0,9,11,106258,701,0.00000010,ODU,OCU,NearRIC
0,3,8,87373,723,0.00000010,ODU,OCU,NearRIC
0,8,10,91373,746,0.00000010,ODU,OCU,NearRIC
0,0,4,70775,746,0.00000010,ODU,OCU,NearRIC
0,5,7,112426,701,0.00000010,ODU,OCU,NearRIC
0,0,11,97440,733,0.00000010,ODU,OCU,NearRIC
0,3,9,111584,726,0.00000010,ODU,OCU,NearRIC
0,1,3,90306,718,0.00000010,ODU,OCU,NearRIC
0,2,10,76683,731,0.00000010,ODU,OCU,NearRIC
```

B.2 ILP Output File

Sample of the VNF instances' provisioning on pseudo-VNFs:

```
cplex5-pz.h:1189 Traffic 0 node 0 provisioned on middlebox 8

cplex5-pz.h:1189 Traffic 0 node 1 provisioned on middlebox 43

cplex5-pz.h:1189 Traffic 0 node 2 provisioned on middlebox 54

cplex5-pz.h:1189 Traffic 0 node 3 provisioned on middlebox 61

cplex5-pz.h:1189 Traffic 0 node 4 provisioned on middlebox 21

cplex5-pz.h:1189 Traffic 1 node 0 provisioned on middlebox 0

cplex5-pz.h:1189 Traffic 1 node 1 provisioned on middlebox 43
```

Sample of identifying the provisioned VNF instances:

```
cplex5-pz.h:1221 [vnfloc] mbox ODU; node 2

cplex5-pz.h:1216 Middlebox 27 (ODU) is active on switch 3

cplex5-pz.h:1221 [vnfloc] mbox ODU; node 4

cplex5-pz.h:1216 Middlebox 28 (ODU) is active on switch 3

cplex5-pz.h:1221 [vnfloc] mbox ODU; node 4

cplex5-pz.h:1216 Middlebox 34 (ODU) is active on switch 5

cplex5-pz.h:1221 [vnfloc] mbox ODU; node 6

cplex5-pz.h:1216 Middlebox 37 (ODU) is active on switch 9

cplex5-pz.h:1221 [vnfloc] mbox ODU; node 10
```

Sample of mapping the traffic links onto physical links in the network:

```
cplex5-pz.h:1281 Traffic 0 link (0, 1) mapped to phy. link (8, 11)
cplex5-pz.h:1281 Traffic 0 link (1, 2) mapped to phy. link (2, 5)
cplex5-pz.h:1281 Traffic 0 link (1, 2) mapped to phy. link (4, 7)
cplex5-pz.h:1281 Traffic 0 link (1, 2) mapped to phy. link (5, 6)
cplex5-pz.h:1281 Traffic 0 link (1, 2) mapped to phy. link (6, 4)
cplex5-pz.h:1281 Traffic 0 link (1, 2) mapped to phy. link (7, 9)
cplex5-pz.h:1281 Traffic 0 link (1, 2) mapped to phy. link (8, 2)
```

B.3 Extracted Features

narrivals Number of arrivals during aggregation interval.

ndepartures Number of departures during aggregation interval (departure occurs at t = arrivaltime + duration).

meanarrrate Mean arrival rate, i.e., number of arrivals / duration of aggreagtion interval.

meandeprate Mean departure rate - cf. arrival rate.

changeN Change in the number of instantiated VNFs (type-independent), i.e., n_vnfs at the end of the aggregation interval - n_vnfs at its beginning.

changeB Change in the total requested bandwidth (type-independent). Unit: kbps.

changeN*VNF* Change in the number of instantiated VNFs (type-dependent).

changeB*VNF* Change in the total requested bandwidth (type-dependent). Unit: kbps.

lastaction*VNF* Last action that was not "doNothing" performed for VNF (type-dependent). Either positive or negative, indicating that the action was "add" / "remove".

n*VNF* Number of VNF instances (type-dependent).

bw*VNF* Amount of requested bandwidth (type-dependent). Unit: kbps.

bwalloc*VNF* Amount of allocated bandwidth (type-dependent). Unit: kbps. This is calculated by multiplying the number of active VNF instances with the VNF capacity.

bwtocap*VNF* Difference between the amount of allocated and requested bandwidth (type-dependent).

bwtoshut*VNF* Difference between the requested amount of bandwidth and the bandwidth level that corresponds to allocating one instance less than is currently active.

util*VNF*: VNF utilization (type-dependent). Utilization is calculated as ratio between requested and allocation bandwidth.

nvnfs	Number of VNFs (type-independent).
totalbw	Total amount of requested bandwidth (type-independent).
mean.nvnfs	Mean number of VNF instances during the aggregation interval (type-independent).
median.nvnfs	Median of the number of VNF instances during the aggregation interval (type-independent).
sd.nvnfs	Standard deviation of the number of VNF instances during the aggregation interval (type-independent).
cv.nvnfs	Coefficient of variation of the number of VNF instances during the aggregation interval (type-independent).
min.nvnfs	Minimum number of VNF instances during the aggregation interval (type-independent).
max.nvnfs	Maximum number of VNF instances during the aggregation interval (type-independent).
slope.nvnfs	"Slope" of . This is calculated as the difference between the mean value in the second half of the aggregation interval and the mean value in the first one.
mean.totalbw	Mean amount of requested bandwidth during the aggregation interval (type-independent).