UNIVERSITY OF TRENTO

Autonomic Management and Orchestration Strategies in MEC-Enabled 5G Networks

Author:
Tejas SUBRAMANYA

Advisor:
Dr. Marco PISTORE

Co-Advisor:
Dr. Roberto RIGGIO

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

School of Information and Communication Technology

January 31st 2021

Declaration of Authorship

I, Tejas SUBRAMANYA, declare that this thesis titled, "Autonomic Management and Orchestration Strategies in MEC-Enabled 5G Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: Tejas Subramanya

Date: 31st January 2021

Acknowledgements

First and foremost, I want to sincerely thank my advisor Dr Marco Pistore and co-advisor Dr Roberto Riggio. I appreciate all their contributions of time, ideas and funding to make my PhD experience productive and stimulating. Their encouragement, guidance and constant feedback have been valuable for my PhD work and my professional development.

I would also like to extend my gratitude to Dr Shuaib Siddique and Dr Emmanouil Kafetzakis for thoroughly reading and providing valuable review comments on my dissertation.

My sincere thanks also go to Dr Henning Sanneck, who offered me an opportunity to join his team as an intern, which unfortunately did not materialize due to the pandemic. I would also like to thank Dr Tarja Hiltunen for her support during this period.

I would also like to extend my appreciation to all the partners in the Horizon 2020 projects - 5GZORRO, 5G-ESSENCE and 5G-CARMEN, for insightful technical discussions over the years, which have been very valuable in my professional development.

I would also like to thank all my colleagues from Fondazione Bruno Kessler and other co-authors for their insights, stimulating discussions and feedbacks throughout my PhD. I am particularly grateful to Dr Tinku Rasheed for encouraging me and supporting me to start my PhD program 4 years ago.

Finally, I would like to extend my deepest gratitude to my parents, Subramanya Nagaraja Shetty and Lakshmi Subramanya, and my wife Harshitha Badrinath, without whose love, support and understanding I could never have completed this doctoral degree.

Abstract

5G and beyond mobile network technology promises to deliver unprecedented ultra-low latency and high data rates, paving the way for many novel applications and services. Network Function Virtualization (NFV) and Multi-access Edge Computing (MEC) are two technologies expected to play a vital role in achieving ambitious Quality of Service requirements of such applications. While NFV provides flexibility by enabling network functions to be dynamically deployed and inter-connected to realize Service Function Chains (SFC), MEC brings the computing capability to the mobile network's edges, thus reducing latency and alleviating the transport network load. However, adequate mechanisms are needed to meet the dynamically changing network service demands (i.e., in single and multiple domains) and optimally utilize the network resources while ensuring that the end-to-end latency requirement of services is always satisfied. In this dissertation work, we break the problem into three separate stages and present the solutions for each one of them.

Firstly, we apply Artificial Intelligence (AI) techniques to drive NFV resource orchestration in MEC-enabled 5G architectures for single and multi-domain scenarios. We propose three deep learning approaches to perform horizontal and vertical Virtual Network Function (VNF) auto-scaling: (i) Multilayer Perceptron (MLP) classification and regression (single-domain), (ii) Centralized Artificial Neural Network (ANN), centralized Long-Short Term Memory (LSTM) and centralized Convolutional Neural Network-LSTM (CNN-LSTM) (single-domain), and (iii) Federated ANN, federated LSTM and federated CNN-LSTM (multi-domain). We evaluate the performance of each of these deep learning models trained over a commercial network operator dataset and investigate the pros and cons of different approaches for VNF auto-scaling. For the first approach, our results show that both MLP classifier and MLP regressor models have strong predicting capability for auto-scaling. However, MLP regressor outperforms MLP classifier in terms of accuracy. For the second approach (one-step prediction), CNN-LSTM performs the best for the QoS-prioritized objective and LSTM performs the best for the

cost-prioritized objective. For the second approach (multi-step prediction), the encoder-decoder CNN-LSTM model outperforms the encoder-decoder LSTM model for both QoS and Cost prioritized objectives. For the third approach, both federated LSTM and federated CNN-LSTM models perform equally better than the federated ANN model. It was also noted that in general federated learning approaches performs poorly compared to centralized learning approaches.

Secondly, we employ Integer Linear Programming (ILP) techniques to formulate and solve a joint user association and SFC placement problem, where each SFC represents a service requested by a user with end-to-end latency and data rate requirements. We also develop a comprehensive end-to-end latency model considering radio delay, backhaul network delay and SFC processing delay for 5G mobile networks. We evaluated the proposed model using simulations based on real-operator network topology and real-world latency values. Our results show that the average end-to-end latency reduces significantly when SFCs are placed at the ME hosts according to their latency and data rate demands. Furthermore, we propose an heuristic algorithm to address the issue of scalability in ILP, that can solve the above association/mapping problem in seconds rather than hours.

Finally, we introduce lightMEC - a lightweight MEC platform for deploying mobile edge computing functionalities which allows hosting of low-latency and bandwidth-intensive applications at the network edge. Measurements conducted over a real-life test demonstrated that lightMEC could actually support practical MEC applications without requiring any change to existing mobile network nodes' functionality in the access and core network segments. The significant benefits of adopting the proposed architecture are analyzed based on a proof-of-concept demonstration of the content caching use case. Furthermore, we introduce the AI-driven Kubernetes orchestration prototype that we implemented by leveraging the lightMEC platform and assess the performance of the proposed deep learning models (from stage 1) in an experimental setup. The prototype evaluations confirm the simulation results achieved in stage 1 of the thesis.

Keywords: 5G, MEC, NFV, Deep Learning, Federated Learning, VNF Scaling, Service Function Chain Placement, Kubernetes, Multi-domain

List of Publications

Publication 1. Subramanya, Tejas, Roberto Riggio. "Centralized and Federated Learning for Predictive VNF Autoscaling in Multi-domain 5G Networks and Beyond" In 2021 IEEE Transactions on Network and Service Management (TNSM), IEEE, 2021.

Publication 2. Carrozzo, Gino, M. Shuaib Siddiqui, August Betzler, José Bonnet, Gregorio Martinez Perez, Aurora Ramos, and Tejas Subramanya. "AI-driven Zero-touch Operations, Security and Trust in Multi-operator 5G Networks: a Conceptual Architecture." In 2020 European Conference on Networks and Communications (EuCNC), pp. 254-258. IEEE, 2020.

Publication 3. Subramanya, Tejas, Davit Harutyunyan, and Roberto Riggio. "Machine learning-driven service function chain placement and scaling in MEC-enabled 5G networks." Computer Networks 166 (2020): 106980.

Publication 4. Subramanya, Tejas, and Roberto Riggio. "Machine learning-driven scaling and placement of virtual network functions at the network edges." In 2019 IEEE Conference on Network Softwarization (NetSoft), pp. 414-422. IEEE, 2019.

Publication 5. Mafakheri, Babak, Tejas Subramanya, Leonardo Goratti, and Roberto Riggio. "Blockchain-based infrastructure sharing in 5G small cell networks." In 2018 14th International Conference on Network and Service Management (CNSM), pp. 313-317. IEEE, 2018.

Publication 6. Subramanya, Tejas, Giovanni Baggio, and Roberto Riggio. "lightmec: A vendor-agnostic platform for multi-access edge computing." In 2018 14th International Conference on Network and Service Management (CNSM), pp. 198-204. ieee, 2018.

Publication 7. Riggio, Roberto, Shah Nawaz Khan, Tejas Subramanya, Imen Grida Ben Yahia, and Diego Lopez. "LightMANO: Converging NFV and

SDN at the Edges of the Network." In NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, pp. 1-9. IEEE, 2018.

Publication 8. Subramanya, Tejas, Leonardo Goratti, Shah Nawaz Khan, Emmanouil Kafetzakis, Ioannis Giannoulakis, and Roberto Riggio. "SDEC: A platform for software defined mobile edge computing research and experimentation." In 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1-2. IEEE, 2017.

Publication 9. Subramanya, Tejas, Leonardo Goratti, Shah Nawaz Khan, Emmanouil Kafetzakis, Ioannis Giannoulakis, and Roberto Riggio. "A practical architecture for mobile edge computing." In 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1-4. IEEE, 2017.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The mobile network technology has been going through a revolutionary change every ten years, with each new generation of technology providing significant performance improvements. Up until the 4*th* generation of mobile technology (4G), these rapid advances have mainly been in response to the capacity demands emerging from the massive data growth over the years, posed mainly by mobile video consumption. However, the usage patterns of the 5*th* generation of mobile technology (5G) and beyond are not just limited to mobile broadband and are envisaged to support use cases with diverse requirements [1]. At the time of writing this thesis (January 2021), 5G standards are slowly approaching their maturity with early 5G deployments beginning worldwide. Nevertheless, there is already a growing interest in 6*th* generation of mobile technology (6G) research (targeting 2030), mainly by telecom operators, vendors, phone manufacturers, governments, and academia [2] [3].

Over the next decade, 5G and beyond are expected to enable new services, reinvent entire industries with new use cases and business models (i.e., industry 4.0), and empower new user experiences by delivering high data rates, extremely low-latency, high reliability, high availability and massive connectivity [4]. In fact, the International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) classifies the diverse 5G use case requirements into three broad categories: (i) enhanced Mobile Broadband (eMBB) which provides greater data-bandwidth complemented by moderate latency improvements (e.g., for high-definition 360-degree video streaming, augmented reality, virtual reality), (ii) massive Machine Type Communication (mMTC) which provides low energy consumption for small data transmissions among a large number of devices (e.g., for remote equipment monitoring, sensors), and (iii) Ultra Reliable Low Latency Communications (URLLC) which provides extremely low-latency, high reliability and high availability (e.g., for remote medical

surgery, connected cars, industrial manufacturing) [5].

To support the services mentioned above over a shared physical network, Communication Service Providers (CSPs) envision a programmable, flexible, scalable, and multi-purpose network, including dynamic and autonomous management of network resources and services. To realize such a vision, CSPs have identified several key technology enablers such as Software-Defined Networking (SDN), Network Function Virtualization (NFV), Multi-Access Edge Computing (MEC), network slicing and Artificial Intelligence (AI), including enhancements to the Radio Access Network (RAN) and 5G Core Network (5GC) protocol stacks [6]. Below we describe each of these technology enablers in detail and also introduce the active and relevant Standard Development Organizations (SDOs) for each of them.

SDN [7] technology is an approach to network management that facilitates CSPs to dynamically and programmatically perform network configuration, thus improving network performance and monitoring. Due to the fact that the static architecture of traditional networks is complex and decentralized, while in contrast, 5G networks require more flexibility and easy troubleshooting, SDN attempts to centralize network intelligence in a single network component by decoupling the control plane (i.e., carrying control and management traffic accountable for configuring the data plane functions) from the data plane (i.e., carries user-generated traffic). The control plane comprises one or more controllers, which are considered the SDN network's brain where the real intelligence exists. ITU-T [8],Internet Engineering Task Force (IETF) [9], Internet Research Task Force (IRTF) [10], Broadband Forum (BBF) [11] and Metro Ethernet Forum (MEF) [12] are some of the SDOs active in the field of SDN.

NFV [13] technology is an approach to network management that offers flexibility and programmability to CSPs by allowing individual network functions to be virtualized (Virtual Network Functions (VNFs)) that could be quickly and efficiently deployed, scaled and migrated based on demand, including chaining the VNFs together to create communication services. NFV relies upon but varies from, traditional server-virtualization techniques, such as those used in enterprise Information Technology (IT). A VNF may consist of one or more virtual machines or containers running different software and processes, on top of standard servers or even cloud computing infrastructure, instead of having custom hardware devices for each network function. IETF, European Telecommunications Standards Institute (ETSI) [14]

and MEF are some of the SDOs active in the field of NFV.

MEC [15] technology is an approach to new network architecture concept that brings IT service environment and cloud computing capabilities such as compute, storage, and networking to the edges of the mobile network. It is designed to rapidly deploy new MEC applications and services at the cellular base stations or edge nodes, thus reducing the delay experienced by end-users and alleviating the transport network load. Moreover, MEC also enables CSPs to open their RAN to approved third parties, such as application developers and content providers. ETSI SDO is the driving force behind the technology of MEC.

Network slicing [16] technology leverages SDN, NFV and MEC to facilitate the coexistence of the earlier mentioned 5G network services. In particular, a network slice can be defined as an independent end-to-end logical network, composed of several chained VNFs that operate on shared physical network infrastructure and tailored to a particular network service requirement. It can span across the entire network, deployed across multiple operators, comprised of dedicated and/or shared resources. 3rd Generation Partnership Project (3GPP) [17] and ETSI are the two SDOs active in the field of network slicing. It is to be noted, that we do not cover network slicing in this thesis work.

Although the technologies discussed above advance mobile networks to evolve into flexible, programmable, scalable and dynamic networks supporting 5G services, they also bring several new challenges, especially in the end-to-end management (i.e., both resource and service management) of such transformed networks [18]. To name a few of the challenges: (i) High dynamicity of network service requirements, (ii) Highly distributed nature of edge-cloud network functions, (iii) Resource-constrained MEC nodes, and (iv) Multi-domain or multi-operator or multi-technology nature of network services.

Two ETSI Industry Specification Groups (ISGs), i.e., ETSI NFV [19] and ETSI Zero-touch Network and Service Management (ZSM), are actively addressing these challenges by introducing new set of management and orchestration functions, together with AI-enabled zero-touch capabilities, to the traditional model of operations, administration, maintenance and provisioning. In this thesis, we address some of these challenges, in line with that of ETSI NFV and ETSI ZSM ISGs, by leveraging optimization techniques (i.e., linear programming) and data-driven AI techniques (i.e., Machine Learning (ML) [20] and Deep Learning (DL) [21]).

1.1 Problem Statement and Methodologies

This thesis is divided into three main problems, and we address each of them individually. This subsection defines all three problem statements and presents an overview of the methodologies used to address those challenges.

1.1.1 AI-driven VNF Auto-scaling in 5G Networks

Management and orchestration of VNFs include not only the traditional Fault, Configuration, Accounting, Performance, and Security Management (FCAPS) but also the lifecycle management of the virtualized resources required for the VNFs. The operations of VNF lifecycle management includes instantiating a VNF, scaling a VNF, updating a VNF and terminating a VNF [22]. In this thesis, we focus on VNF scaling, which may include changing the configuration of the virtualized resources (i.e., scale-up by adding a CPU or scale-down by removing a CPU), adding new VNF instances (i.e., scale-out by adding a new virtual machine), removing existing VNF instances (i.e., scale-in by removing a virtual machine). The scale-in and scale-out operations are characterized into horizontal scaling while the scale-down and scale-up operations are categorized into vertical scaling. ***Traditionally, VNF autoscaling has been reactive in nature, but with increasing network service dynamicity in 5G and beyond networks, there needs to be a mechanism for CSPs to proactively scale VNFs by dynamically adapting to the network changes.***

The ZSM framework architecture proposed by ETSI ZSM ISG supports network monitoring and data collection across multiple network management domains. Therefore, to realize proactive VNF auto-scaling, there is a need for data-driven automation based on closed-loop and integration of AI techniques. For example, ML or DL can be used to predict the network load requirements for a future time instance and accordingly proactively scale the VNFs. However, sometimes this might require aggregating operational data from various data sources (e.g., MEC nodes) belonging to multiple network management domains (e.g., CSPs), for insightful analytics. Nevertheless, such aggregation mechanisms incur practical challenges, such as regulatory restrictions on sharing sensitive user data (e.g., General Data Protection Regulation), high bandwidth resources required to transfer the data to a central aggregator and high risk associated with a single point of failure. Notably, in a multi-CSP ecosystem, assuring

isolation among CSPs is crucial because of the security concerns associated with one CSP possessing access to other CSP data. Therefore, to better react to the changing network service requirements, optimize resource usage, and comply with data privacy and protection policies, there is a need for data to be processed and analyzed in a distributed manner. In such a situation, distributed learning algorithms [23] can be used to train ML or DL models that perform proactive auto-scaling from data distributed across multiple data silos, eliminating the need for raw data sharing in a multi-CSP ecosystem.

In this thesis, we propose two centralized DL approaches for VNF auto-scaling, which include Multilayer Perceptron (MLP)-based classification and regression [24], and Deep Neural Network (DNN)-based (i.e., Artificial Neural Network (ANN), Long Short-term Memory (LSTM) and Convolutional Neural Network–Long Short-term Memory (CNN-LSTM)) Time Series Forecasting [25]. Additionally, we also propose encoder-decoder LSTM and CNN-LSTM models for multi-step Time Series Forecasting in VNF autoscaling. Most importantly, we propose a distributed learning approach called Federated DL [26] for VNF auto-scaling. In contrast to existing works (to be described in Chapter 2) on proactive VNF auto-scaling, we use real-operator traffic traces to generate training datasets required for predicting VNF auto-scaling decisions, unlike other works that are based on simulated datasets. Moreover, we are also the first to propose a Federated Learning (FL) approach for VNF auto-scaling in multi-CSP networks.

1.1.2 Service Function Chain Placement in 5G Networks

Individual VNFs can be chained together (e.g., VNF Forwarding Graph (VNFFG)) to form a Service Function Chain (SFC) that represents a specific network service with a guaranteed latency and data rate requirements, as requested by the User Equipments (UEs) [22]. There exist multiple locations of Network Function Virtualization Infrastructure (NFVI) Point-of-presence (PoP) (e.g., base station, aggregation point, 5GC) for instantiating VNFs, which can be shared by many UEs. Management and orchestration of such network services include creating, updating and deleting VNFFG associated with a network service. ***With the distributed nature of edge-cloud network functions in 5G networks and beyond, there needs to be a mechanism to associate UEs to base stations efficiently and to embed VNFs onto the substrate network by allocating radio, compute and***

transport resources to network services based on the objective defined by the CSP.

The NFV framework architecture proposed by ETSI NFV ISG enables CSPs to share their network infrastructure with other third parties (e.g., Mobile Virtual Network Operator (MVNO)) by allowing them to make virtual network requests that include their desired network services and Quality of Service (QoS) expectations. The problem of embedding these virtual network requests onto a shared network infrastructure has been proven to be NP-hard and has been extensively studied in the literature [27]. The embedding process involves two steps: node embedding and link embedding. In the node embedding step, each VNF in the request should be mapped to a substrate node, while in the link embedding step, each virtual link in the request should be mapped to a path in the substrate network.

In this thesis, we formulate a virtual network embedding problem for optimally solving user association and SFC placement in 5G networks based on mathematical optimization techniques and executed using the CPLEX optimization solver [28]. These optimization models have an objective function that either minimizes or maximizes a particular cost function by satisfying all the constraints. Although the mathematical optimization techniques always achieve an optimal solution to the formulated problem, they are not scalable for larger problem size. Therefore, we further propose a heuristic algorithm that reaches a near-optimal solution in a much shorter time frame. In contrast to the existing works on SFC placement problems, we consider the joint problem of user association and SFC placement which allows the optimization of end-to-end latency according to user locations, SFC latency and data rate requirements, and computing/networking resource availabilities. Furthermore, our proposed latency model stands out from the existing delay models within the context of 5G mobile networks.

1.1.3 AI-driven MEC Orchestration Platform

With the acceptance of SDN, NFV and MEC as technology enablers for 5G and beyond, network transformation is currently occurring rapidly within the infrastructure of a network provider leading to new requirements on the way networks are deployed and managed. One major challenge with MEC is driving the lifecycle management and orchestration of VNFs or MEC applications hosted on 'resource-constrained' MEC nodes at the edges of the network. ***Most of the work in this area are either based on simulations and***

analytical evaluations that validate simplistic scenarios (will be discussed in Chapter 2).

In this thesis, we take a two-step approach to illustrate our implemented platform. First, we introduce 'lightMEC' platform - a lightweight solution (e.g., leveraging container virtualization technology [29]) to deploy mobile edge computing functionalities that allows hosting of low-latency and bandwidth-intensive applications at the network edges. Measurements conducted over a real-life test demonstrated that lightMEC could support practical MEC applications without changing existing mobile network nodes' functionality in the access and core network segments. The significant benefits of adopting the proposed architecture are analyzed based on a proof-of-concept demonstration of the content caching use case. Second, we extend the Kubernetes [30] architecture to introduce Monitor, Analyze, Plan, and Execute (MAPE) closed control loop that enables AI-driven lifecycle management of MEC applications hosted on the lightMEC platform. Measurements conducted over Kubernetes test-bed demonstrated that proactive auto-scaling improves the performance of MEC applications.

1.2 Thesis Organization

The structure of this thesis is summarized as follows. In the current chapter, we first described our motivation and objectives of the thesis. The problem statements, together with the approaches undertaken to solve those problems, were discussed in brief.

Chapter 2 will provide an in-depth understanding of 5g mobile networks, MEC, NFV Management and Orchestration principles, AI and ML. Then, we present related works on VNF auto-scaling, VNF and SFC placement, user association and MEC orchestration platforms.

Chapter 3 will define the problem statement for VNF autoscaling in centralized and distributed networks. Then, we propose three different approaches for proactive VNF auto-scaling leveraging centralized and federated DL techniques. For each method, we provide details on the dataset used, data transformation and feature engineering, ML model architecture, and the relevant performance metrics to evaluate the results.

Chapter 4 discusses the joint problem of user association and SFC placement in MEC-enabled 5G networks. In particular, we present 5G mobile network model, SFC request model, UE association, scheduling and delay

model. We then formulate the optimization problem using Integer Linear Programming (ILP) technique and compare the optimal solution with a heuristic solution with various simulations.

Chapter 5 presents the system design and architecture for both lightMEC and AI-driven MEC orchestration platforms. Moreover, we provide prototype details for both the platforms and illustrate their benefits through performance evaluations.

Chapter 6 summarizes this thesis's main contributions, followed by highlighting the key findings of the work. Moreover, several promising research directions for future works will be presented.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 5G Mobile Network

At a very high level, Fig. 2.1 represents the 5G mobile network architecture. A 5G mobile network is composed of two major elements: the 5G Access Network and the 5GC. The 5G Access Network comprises one logical node, the next-generation NodeB (gNodeB), which connects to UEs, providing control plane and user plane services. The gNodeBs are interconnected to each other using the Xn interface. The 5GC consists of many logical nodes such as Access and Mobility Management Function (AMF), Session Management Function (SMF), and User Plane Function (UPF). The gNodeBs are connected to the 5GC through NG interfaces, more specifically to the AMF through NG-C control plane interface and to the UPF1 through NG-U user plane interface. The components within the 5GC are also interconnected using standardized interfaces.

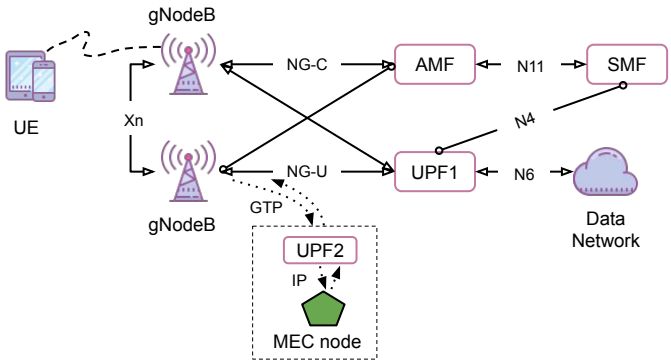


FIGURE 2.1: 5G Mobile Network Overview.

The scheduler entity in the gNodeB is responsible for deciding which UEs should be allocated air interface resources, i.e., Physical Resource Blocks (PRBs) on Transmission Time Interval (TTI) (e.g., $1ms$ or $0.5ms$ or

0.25*ms* or 0.125*ms*) basis and how much PRBs should be allocated to send or receive data. Once a UE attaches to the network using control plane signaling (e.g., attach process), it can send/receive data to/from the Packet Data Network (PDN) using the GPRS Tunnelling Protocol (GTP). The uplink UE traffic received by the gNodeB over its air interface is encapsulated into a GTP packet and then delivered to the UPF1 over the NG-U interface. This GTP tunnel is terminated at the UPF1 which removes the GTP header and forwards the UE traffic to its intended destination (e.g., the Internet).

Besides, consider a MEC node being placed in between the gNodeB and the 5GC to support low-latency applications. For the MEC applications to operate, it needs to have access to the UE IP traffic. Therefore, the integration of MEC in the 5G network requires the co-location of a UPF network function with the MEC node. The UPF takes care of performing a stateful termination and recreation of the GTP session concerning the UE. As we see in Fig. 2.1, the GTP-encapsulated UE traffic is redirected from the gNodeB to the UPF2. Here the GTP tunnel is terminated, and the UE IP traffic, now accessible, is redirected to the MEC node running MEC applications. On the way back, the GTP tunnel is recreated by UPF2, and the response is delivered back to the UE.

2.1.2 Cloud Computing vs Multi-access Edge Computing

Cloud Computing [31] is a network paradigm that refers to both the applications delivered as services over the mobile network as well as the hardware and systems software within the data centres that offer those services. A public cloud (e.g., Amazon's Elastic Cloud, Microsoft's Azure platform) refers to a cluster of computer hardware and software that offers the general public services on a pay-per-use basis. On the other hand, a private cloud refers to data centres internal to an organization.

Virtualization of resources is an essential requirement for any cloud provider. It is necessary for the cloud's resource scalability to meet the varying requests from the cloud user and to obtain and release resources dynamically with minimum management efforts. Many applications and services that utilize AI and big data analytics are tailored to a cloud-based approach for data processing. Despite the advantages of cloud computing, there exist several challenges for emerging low-latency applications (e.g., virtual reality, connected cars) such as long-distance between the data

source and data consumer and massive data transmission to the cloud leading to excessive end-to-end delay and backhaul resource depletion.

The origin of MEC is based on many complementary technologies, including cloud computing, mobile cloud computing [32], fog computing [33] and cloudlet [34]. The main idea of MEC is to provide content providers and application developers with an IT service environment and cloud-computing capabilities at the edges of the mobile network, within the RAN and in proximity to mobile subscribers. Therefore, MEC enables a new ecosystem and value chain, i.e., Mobile Network Operators (MNOs) can open their RAN to authorized third-parties, permitting them to flexibly and quickly deploy innovative applications and services towards mobile subscribers, vertical segments, and enterprises.

The demand for MEC is driven by various factors, such as the introduction of new mobile technologies (i.e., 5G, 5G+, 6G) requiring high-bandwidth and low-latency applications (e.g., virtual reality, interactive gaming, mission-critical controls), the accelerated rise in data generation within the mobile network, and advancements in the field of computing and artificial intelligence. However, even though MEC can function in a standalone environment, a centralized cloud and distributed MEC complement each other exceptionally well. Such coexistence of centralized cloud and distributed MEC will play an essential role in various situations since the resource bottleneck of MEC servers has become more prominent due to the accelerated increase in the number of connected devices, data volume, and analytics.

2.1.3 NFV Management and Orchestration

Guaranteeing a highly available framework that can perform end-to-end service provisioning demands an effective system to monitor, manage, control and orchestrate network functions and services. In this regard, the standardization bodies have devoted a significant effort to develop an NFV Management and Orchestration (MANO) framework. For example, the ETSI standardization body has recognized the potential of NFV in future mobile networks and hence formed an ISG to develop a prevalent NFV framework.

The ETSI MANO reference model defines a set of logical components and their interfaces. The framework consists of two main stacks: the NFVI stack and the MANO stack. The NFVI stack includes both the physical and virtualized resources required to host VNFs on a logically isolated shared

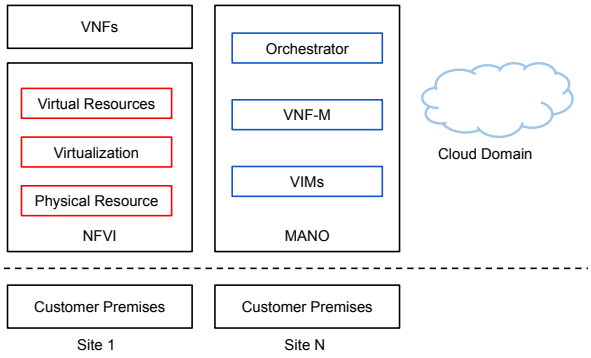


FIGURE 2.2: The ETSI MANO Reference Architecture.

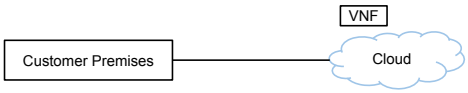
infrastructure. The VNFs are software implementation of legacy network functions. Conversely, the MANO stack hosts the service and function orchestration and lifecycle management logic (i.e., NFV Orchestrator (NFVO) and Virtual Network Function Manager (VNF)) as well as the Virtual Infrastructure Manager (VIM) in charge of controlling the physical and virtual resources available in the NFVI PoP. The reference ETSI MANO model is sketched in Fig. 2.2. The service lifecycle management comprises of operations such as instantiation, placement, scaling, migration, and termination of VNF instances to guarantee service continuity throughout the runtime of the VNFs.

The scenario depicted above does not specify how the actual NFV platform must be implemented, nevertheless the flexible service provisioning requirements and the heavy reliance on virtualization led many implementers to reuse software stacks and best practises found in the data-center and cloud computing domains. This despite the fact that, being software instances, VNFs can essentially run on any kind of platform supporting NFVI.

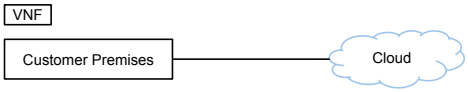
Figure 2.3 depicts three NFV deployment models. In the centralized NFV deployment model all VNFs are deployed in large data-centers. In the distributed NFV deployment model, VNFs runs on MEC platform deployed at the customer premises. Finally, in the hybrid model VNFs are deployed both at the edge of the network and in the centralized cloud.

2.1.4 Artificial Intelligence and Machine Learning

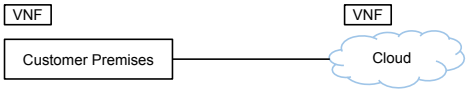
AI enables machines to learn from experience, adjust to new inputs and perform human-like tasks. Most AI examples rely heavily on ML and DL which trains machines to accomplish precise tasks by processing large



(A) Centralized deployment model.



(B) Distributed deployment model.



(C) Hybrid deployment model.

FIGURE 2.3: NFV Deployment models.

volumes of data and identifying patterns in the data. ML or DL are classified into two broad categories: supervised learning and unsupervised learning.

Supervised Learning [35] trains the machine using well-labelled data that means data is already marked with the correct answer. The machine is then provided with a new set of data so that a supervised learning algorithm analyzes the training data and learns the mapping function from the input to the output to produce a correct outcome from labelled data. Supervised learning is divided into two classes of algorithms:

Classification: A classification problem is when the output variable is a category, such as 'Yes' or 'No' or 'increase' and 'decrease'.

Regression: A regression problem is when the output variable is a real value, such as 'dollars' or 'meters'.

Unsupervised Learning [36] trains the machine using information that is neither classified nor labelled and allows the algorithm to act on that knowledge without supervision. Here, the machine's task is to group unsorted information according to similarities, patterns, and differences without any prior training of data. Unsupervised learning is divided into

two classes of algorithms:

Clustering: A clustering problem is when you want to discover the inherent groupings in the data.

Association: A association problem is where you want to discover rules that describe large portions of your data.

Deep Learning or DNN is a subset of ML in AI that has networks capable of imitating the workings of the human brain in processing raw data and learning patterns for effective decision making. ANN, Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN) are the three most common types of DNNs.

ANNs are the simplest type of neural networks that do not have any cyclic connections, and the data passes from the input layer to the output layer in a single pass without any state memory of what arrived before. A simple example of the ANN is the MLP.

CNNs are the regularized versions of MLPs that have shared-weights architecture and spatial-invariance characteristics to learn local patterns efficiently, mostly, in images. CNNs contain one or more convolutional layers, which can either be wholly interconnected or pooled. Since the convolutional layer uses a convolutional operation on the input, the network can be deeper with only a few parameters.

RNNs are networks that have cycles and include state memory to process sequences of inputs. RNNs share weights across time, unlike CNNs, which share weights across space, thus enabling them to process and represent patterns in sequential data efficiently. Most common variants of RNNs include LSTM and Gated Recurrent Units (GRU).

Federated Learning is a recent addition to distributed ML, which aims at training a ML or DL algorithm, across multiple local datasets, contained in decentralized edge devices or servers holding local data samples, without exchanging their data — thus addressing critical issues such as data privacy, data security, and data access rights to heterogeneous data. This approach of FL is in contrast to traditional centralized learning techniques where all data samples are forwarded to a centralized server and also to classical distributed ML techniques, which assume that the local data samples are identically distributed and have the same size.

The general design of FL involves training local models on local data samples and exchanging parameters (e.g., weights in a DNN) among those local models to generate a global model. FL algorithms can use a centralized server that orchestrates the various steps of the algorithm and serves as a

reference clock, or they may be peer-to-peer, where no centralized server exists [26].

The FL process is divided into multiple rounds, each consisting of four steps:

Step 1: Local training - All local servers compute training gradients or parameters and send locally trained model parameters to the central server.

Step 2: Model aggregating - The central server performs secure aggregation of the uploaded parameters from n local servers without learning any local information.

Step 3: Parameters broadcasting - The central server broadcasts the aggregated parameters to the n local servers.

Step 4: Model updating - All local servers update their respective models with the received aggregated parameters and examines the performance of updated models.

After several local training and update exchanges between the central server and its associated local servers, it is possible to achieve a global optimal learning model.

2.2 Related Work

2.2.1 VNF Auto-scaling

One of the significant hurdles for the deployment of NFV is the resource allocation of demanded VNFs or network services (i.e., a chain of VNFs) in NFV-based network infrastructures. A comprehensive state of the art on NFV resource allocation is discussed in [37] by presenting the fundamental research challenges and introducing a classification of the main approaches that pose solutions to solve it. One of the potential solutions for the resource allocation problem also involves VNF auto-scaling [38]. Previous works on VNF auto-scaling can be divided into two categories: reactive mode and proactive mode.

Reactive Auto-scaling. In reactive mode, threshold levels can be either statically pre-defined or dynamically updated. In [39] and [40], the authors propose scalability mechanisms based on static thresholds. They define two threshold levels ($scalein_{thr}$ and $scaleout_{thr}$) to determine if the load reduces below or exceeds above the respective limits and accordingly triggers the scaling process. However, such techniques may result in oscillating behaviour affecting the overall system performance. On the other hand, [41]

and [42] propose mechanisms such as queuing theory and reinforcement learning, which allows the scaling policy to be improved based on dynamic or adaptive thresholds. Although it performs better than static approaches, it remains a reactive solution with similar weaknesses.

Proactive Auto-scaling. In proactive mode, forecasting techniques (e.g., time series forecasting) are applied to allow the systems to learn automatically and to anticipate future needs, based on which scalability decisions are taken.

The field of time series forecasting has been primarily influenced by linear statistical methods such as Moving Average, Auto-Regressive and Auto-Regressive Integrated Moving Average. In the 1980s, it was evident that linear models do not apply to many real-world applications [43]. Therefore, researchers proposed several non-linear time series models such as the bilinear model, the threshold autoregressive model, and the Autoregressive Conditional Heteroskedasticity. We point the readers to [43] and [44] for a detailed review of previously introduced linear and non-linear time series models.

In the last two decades, ML models have become main contenders to classical statistical models within the time series forecasting community. These black-box or data-driven models are examples of nonparametric non-linear models which learns the stochastic dependency between the past and the future using only historical data. [45] provides a detailed review of different ML models (e.g., ANN, decision trees, support vector machines, nearest neighbour regression) for time series forecasting. Furthermore, the empirical comparison of various ML models for time series forecasting is discussed in [46].

In the 5G domain, there already exists some literature on the use of ML models with time series forecasting to predict traffic loads and inturn to perform VNF auto-scaling. For instance, a comprehensive survey on different ML methods for reliable resource provisioning in edge-cloud computing ecosystem is discussed in [47]. The authors in [48] explore DNN-based and LSTM-based forecasting framework for VNF resource requirement prediction using Synthetic Minority Oversampling Technique and Batch Normalization layer to deal with the imbalanced dataset. Moreover, they explore the impact of different feature vector size and the number of hidden layers on prediction accuracy. They also propose and evaluate the performance of hybrid LSTM models such as CNN-LSTM and Bidirectional-LSTM models. Similarly, [49] offers a novel mechanism, using

DNN and LSTM, to scale 5GC virtual functions by forecasting the upcoming traffic load. Through simulations, they compare the performance of proactive VNF auto-scaling to threshold-based reactive auto-scaling.

Similar to the works mentioned above, in our work, we explore FFNN-based, LSTM-based and CNN-LSTM-based traffic load forecasting frameworks for VNF autoscaling. But also, we explore encoder-decoder LSTM and CNN-LSTM models for multi-step time-series forecasting in VNF autoscaling. Moreover, we consider both horizontal and vertical VNF autoscaling, unlike other works that mostly consider only horizontal approach. But most importantly, we also propose using Federated Deep Learning techniques for predictive VNF autoscaling to preserve privacy among various participants (i.e., multi-domains) in the 5G ecosystem.

2.2.2 VNF and SFC Placement

Optimal placement of VNFs to the substrate network is challenging and should be implemented carefully.

There exists a significant amount of literature on placing VNFs in the NFV infrastructure ([50], [51], [52] and [53]). VNF-P [50] can be considered as one of the most prominent works on VNF placement, where the authors present a generic model for efficient placement of VNFs. In [51], the authors determine the required number of VNFs and their optimal placement in such a way that minimizes network operational costs and maximizes network utilization. In [52], the authors present a real-world implementation of VNF placement on OpenStack [54] to optimize the overall system performance and to provision resilience. The authors in [53] formulate the VNF placement problem as a resource-constrained shortest path problem to minimize the overall latency.

There already exists some literature on the SFC placement problem with certain end-to-end latency needs that need to be satisfied [55], [56], and [57]. In [55], the authors present a delay-aware SFCs placement problem such that VNFs forming SFCss are placed so as to satisfy end-to-end latency demands while utilizing network resources in an effective manner. A joint VNF placement and CPU allocation problem is studied in [56] and an optimization problem is formulated by employing a queuing-based model to minimize the ratio between the actual and the maximum allowed latency, for all SFCs requests. The authors in [57] study the problem of VNF instantiation and migration with a goal of minimizing SFCs delays.

However, all of these studies do not consider UE processing time, gNodeB processing time, and propagation or transmission time over the air interface. Besides, none of the above studies consider heterogeneous MEC nodes, which increases the search space causing the SFCs placement problem to grow cumbersome.

2.2.3 MEC Orchestration Platform

MECs aims at converging IT services and telecommunications, providing cloud-computing capabilities to deploy various applications at the edge of the radio access networks. By bringing storage resources, computational resources and applications at the network edges, both service latency and backhaul load can be reduced significantly. MECs also provides real-time access to radio network information that can be leveraged by authorized third parties such as content providers, to develop innovative MECs applications and services targeted towards mobile users, enterprises and vertical segments.

The ETSI is active in the MECs arena with a dedicated Industry Specification Group. The activities of this group are still in their early stages, nevertheless a number of drafts specification are already available. For example, [58] describes the technical requirements of MECs, while its framework and reference architecture are illustrated in [59]. The service scenarios benefiting from MECs and the proof-of-concept implementation details are illustrated in [60] and [61] respectively.

The MECs related research has received a considerable attention in recent times. The authors in [62] provide an architectural blueprint for MECs and discuss the technical advantages it offers by presenting a number of use cases. A taxonomy of MECs along with its key attributes are discussed in [63]. The authors also present some promising real-time MECs application scenarios. The challenges involved in commercial deployment of MECs and the progress towards it are discussed in [64]. In [65] and [66], the authors present different mobile offloading techniques in cellular networks to enable low-latency applications. The authors in [64] introduce a MECs platform called WiCloud that provides edge computing and proximity services for innovative applications. They also discuss on the current progresses and the challenges associated with MEC. In literature, several other MECs architectures have been proposed such as Mobile Micro-cloud [67], MobiScud [68], Follow-Me-Cloud [69] and CONCERT [70].

Most of the MEC architecture work reported above are either based on simulations and analytical evaluations that validate only simplistic MEC scenarios. Conversely, in this paper, we provide a practical implementation of the proposed MEC framework which is then evaluated based on the caching use case.

Chapter 3

AI-driven VNF Auto-scaling in 5G Networks

This chapter proposes various DL models that can assist in proactive VNF auto-scaling by predicting the required number of VNF instances based on traffic traces collected over a real-operator commercial network. Firstly, in Section 3.1 and Section 3.2, we illustrate two Centralized Learning (CL) approaches, i.e, Classification/Regression and Time Series Forecasting, respectively, for VNF auto-scaling within a single domain (e.g., single network operator). Secondly, in Section 3.3, we illustrate FL-based Time Series Forecasting approach for VNF auto-scaling across multiple domains (e.g., multiple network operators). We evaluate the performance of proposed DL models trained over a commercial network operator dataset and investigate the pros and cons of various approaches through extensive simulations. This chapter is based on three of our publications - [71] [72] [73].

3.1 Centralized Deep Learning - Classification and Regression

3.1.1 Overview

This section considers a 5G mobile network, composed of six base stations, two aggregation points, and one 5G Core, as depicted in Fig. 3.1. A set of three base stations are interconnected to each other through Xn-interfaces. Utilizing NG-interfaces, the six base stations are served by two aggregation points, and the 5G Core serves both of these aggregation points. Each element in our network topology is equipped with a resource-constrained (e.g., CPU) Mobile Edge (ME) Host that is capable of hosting VNFs. The

closer the ME hosts are towards the end-users, the scarcer their computational resources become. It is necessary to mention that the integration of ME host into the 5G network requires the collocation of a 5G Core UPF element with each ME host to reap the benefits of the MEC system. The functionality of the UPF in MEC systems is explained in Chapter 2, collectively with other relevant concepts and terminologies of 5G mobile network.

The rationale behind choosing such a 5G network topology is to align with the dataset that we obtained from a commercial network operator consisting of traffic load samples only from six base stations. Moreover, considering the fact that the proactive VNF autoscaling predictions obtained from this section will be used as an input to the ILP model in Chapter 4 to determine the latency-optimal ME Host to initialize the predicted VNF instance(s), ME hosts were introduced in all six base stations, two aggregation points, and the 5G Core so as to increase the complexity of the ILP problem.

In the rest of this section, we examine ANN based MLP Classification and Regression models to predict the required number of Virtual User Plane Function (vUPF) instances as a function of the network traffic they should process in each base station, and evaluate the performance of both these models. Additionally, we study and evaluate QoS-prioritized and Cost-prioritized techniques for both MLP Classification and Regression models. ***It is to be noted that the output from the best performing MLP model, i.e., 'the number of vUPF instances' will be fed as an input to the ILP model in Chapter 4, which determines the ideal ME Host (e.g., based on QoS parameters) to initialize the predicted vUPF instances.***

3.1.2 Problem Statement

Given Dataset: The dataset utilized in this section is generated from a commercial operator by monitoring the mobile network traffic load on six base stations, with each base station having ten cells, for eight consecutive days. The traces in the dataset are in the form of a time series $\{(x_t, y_t)\}$ and we interpret this time series as a set of samples $\{(x_1, y_1)$, $(x_2, y_2), ..., (x_n, y_n)\}$. The traces are collected on an hourly timescale. The samples indicate the average traffic load per second for that hour and not the peak traffic load. So, the samples do not account for any short bursts (e.g., due to unexpected events) in traffic for that hour.

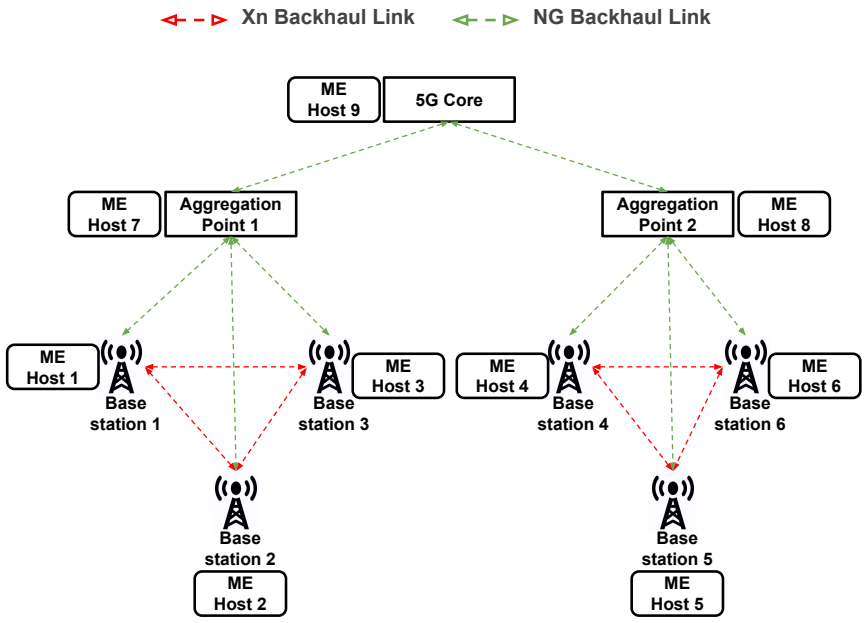


FIGURE 3.1: 5G Mobile Network

Find: We investigate how to map traffic load statistics X to vUPF scaling decisions Y using supervised learning, which involves learning from a training set of data. The details on the composition of X and Y are discussed in Section 3.1.3. The X and Y metrics evolve over time, influenced mainly by the mobile network traffic dynamics and the actual number of mobile users. Our goal is to determine the distribution of scaling decision metric Y constrained on knowing the traffic load metric $x \in X$. Employing the statistical learning framework, we model X and Y as random variables. We assume that each sample (x_t, y_t) in the training set is obtained from the conditional probability distribution of (X, Y). Further, we suppose that x_t is multi-dimensional (multi-variate) and y_t is one-dimensional (uni-variate). In this formalism, the inference problem consists of finding a model $F : x \text{->} P(Y|x)$ for $x \in X$, so as to maximize the likelihood function $L(\{P(y_t|x_t)\})$, which can be attained by minimizing the loss/error function.

Objective: The vUPF scaling decisions Y refer to the required number of vUPF instances to process incoming traffic with an objective to either maximize QoS or minimize cost. The logic behind mapping the traffic load samples in X to the number of vUPF instances required to process that traffic load is based on the above-chosen objective. The accuracy of the prediction largely influences the achieved goal.

3.1.3 Data Transformation and Feature Engineering

We now describe the input feature sets $X_{default}$ and $X_{constructed}$ of the dataset, which, when combined, is referred to as X, as well as the output classes or real-valued quantities Y. The $X_{default}$ feature set includes 8 numeric features that are already available in the dataset, as described in Table 3.1. In addition to these default features, we construct 9 numeric features ($X_{constructed}$) from the basic dataset, as shown in Table 3.2, using a process called *feature transformation* by extending backward from time t. These constructed features contain information or patterns on how the traffic load evolves, therefore assisting in proactive vUPF scaling decisions.

Default features ($X_{default}$)
1. base station ID.
2. Date.
3. Time-stamp t.
4. Average number of users between t and $t-1$ in each cell.
5. Maximum number of users between t and $t-1$ in each cell.
6. Average downlink user throughput in each cell.
7. Average uplink user throughput in each cell.
8. Traffic load measured in each cell at time t, given by $\lambda(t)$.

TABLE 3.1: Default set of features available in the dataset.

The next step is to define how we generate output classes or real-valued quantity Y, which the MLP classifier or regressor tries to predict, respectively. In vUPF auto-scaling, there is a trade-off between QoS and cost. More vUPF instances (i.e., resources) need to be allocated to guarantee QoS, but allocating more resources raises the cost. Therefore, we propose two different approaches: (i) QoS favored classifier/regressor (Q-classifier/Q-regressor) and (ii) Cost favored classifier/regressor (C-classifier/C-regressor).

In Q-classifier/Q-regressor, the network operator gives priority to QoS over the cost. The auto-scaling decision at step n considers future traffic demands until the next auto-scaling step $n+1$. Therefore, the class value or the target variable value is generated as follows:

$$Y_Q = min(vnf_{max}, max(\frac{\lambda(t)}{\gamma}))\forall t \in \{\tau(n), ..., \tau(n+1)\} \tag{3.1}$$

Constructed features ($X_{constructed}$)
9. Traffic load measured in each cell at time $t-1$, given by $\lambda(t-1)$.
10. Traffic load measured in each cell at time $t-2$, given by $\lambda(t-2)$.
11. Traffic load measured in each cell at time $t-3$, given by $\lambda(t-3)$.
12. Traffic load measured in each cell at time $t-4$, given by $\lambda(t-4)$.
13. Change in traffic load in each cell from time t to $t-1$.
14. Change in traffic load in each cell from time $t-1$ to $t-2$.
15. Change in traffic load in each cell from time $t-2$ to $t-3$.
16. Change in traffic load in each cell from time $t-3$ to $t-4$.
17. Weekday or weekend.

TABLE 3.2: Constructed set of features from the dataset.

where t are the timestamps containing traffic data samples between steps n and $n+1$ (including $\tau(n)$ and $\tau(n+1)$), $\lambda(t)$ is the traffic load in a cell at time t, γ is the maximum traffic load a single vUPF can handle, and vnf_{max} is the maximum number of vUPF instances per cell that can be hosted on the ME Host.

In C-classifier/C-regressor, the network operator chooses to neglect short-lived bursty traffic between steps n and $n+1$ to avoid over-provisioning of vUPF instances, therefore minimizing cost and enduring short-lived degradations. Consequently, the auto-scaling decision considers measured traffic load only at step n and at next auto-scaling step $n+1$. Therefore, the class value or the target variable value is generated as follows:

$$Y_C = min(vnf_{max}, max(\frac{\lambda(\tau(n))}{\gamma}, \frac{\lambda(\tau(n+1))}{\gamma})) \tag{3.2}$$

where $\tau(n)$ is the time at which step n occurs and $\tau(n+1)$ is the time at which step $n+1$ occurs.

It is to be noted that in our work, we examine the performance metrics for Q-classifier, Q-regressor, C-classifier, and C-regressor for two cases: (i) auto-scaling decisions performed on *one* hour time-intervals and (ii) auto-scaling decisions performed on *two* hour time-intervals. Considering that

the dataset we obtained from the commercial network operator is on hourly time-scale, the lowest time granularity for training an ML model to perform auto-scaling predictions is *one* hour. Although our traffic traces are collected on hourly time intervals, our model is generic enough to handle lower time interval granularities (e.g., 5-minute time interval data samples) and different auto-scaling steps (e.g., 1 hour, 5 hours).

Next, we identify the dominant features from our feature list based on their influence on classification 'accuracy' or regression 'R-squared' values using Recursive Feature Elimination (RFE) and Principal Component Analysis (PCA) techniques.

RFE is a greedy optimization technique that strives to find the best performing feature subset. It repeatedly generates models and keeps aside the best or the worst performing feature in each iteration. The next model is constructed with the remaining features until all the features are depleted. It then ranks the features based on the order of their elimination. With MLP, it is difficult to understand which input features are relevant and which are not. The reason being, each input feature has multiple coefficients that are linked to it - each corresponding to one node of the first hidden layer. Additional hidden layers make it even more challenging to decide how big of an impact the input feature has on the final prediction. Therefore, we apply the RFE technique on a linear support vector machine model to find the optimal number of features and use them in creating our MLP models. After ranking, features 8, 9, 10, 11, and 12 are ranked highest, which implies that measured loads closer to the scaling decision time are the crucial features. Features 2, 17, and 3 are ranked 2nd, 3rd, and 4th, respectively. The rest of the features are ranked in the following order: 14, 15, 13, and 16. Finally, features 1, 4, 5, 6, and 7 are recommended not to be used in the model (RFE returns 'false').

We have also validated this observation using PCA, a statistical method to find correlated features and their impact on classification and regression. In PCA, the first principal component has the most notable variance, accounting for much of the variability in the data samples. Our PCA lists a combination of features 8, 9, 10, 11, and 12 as the first principal component, indicating similar conclusions as RFE.

Based on the ranking of these features, we use only 12 features (eliminating $1, 4, 5, 6$ and 7 from Table 3.1) that provides the best results for our MLP models.

Once data is collected and features extracted/selected, the dataset is

decomposed into training, validation and test datasets. We use a rule-of-thumb decomposition conforming to 60%/20%/20% between the training, validation and test datasets, respectively. The data samples chosen for training and validation (i.e., close to 6 days of data from 6 base stations) are the most balanced data in our dataset compared to samples from other days that were used for testing (close to 2 days). Even then, the training samples are slightly imbalanced in classes/target real-valued quantities which might result in over-fitting the model (i.e., a condition where a statistical model begins to output a random class or error value outside the original dataset), and therefore we look at other performance metrics such as confusion matrix, precision, recall, and F1-score for MLP classifier and R-squared value for MLP regressor, which can provide more insight into the performance of our MLP models than traditional classification accuracy and regression mean squared error, which are excellent measures only if the datasets are entirely symmetric. Finally, if the performance metrics indicate over-fitting, the K fold cross-validation technique can be used to generate multiple mini train-test splits to tune our MLP models, which was not necessary in our case.

3.1.4 Proposed Centralized Deep Learning Models

In this subsection, we create two types of MLP models, a *classifier* and a *regressor*, that can identify and exploit hidden patterns in network traffic load instances to predict vUPF scaling decisions ahead of time. In particular, we illustrate on different steps involved in creating our models and eventually evaluate them based on several performance metrics [74].

An MLP is a class of feed-forward ANN, consisting of at least three layers of nodes (neurons): an input layer, one or more hidden layers, and an output layer, as shown in Fig. 3.2 and Fig. 3.3. These nodes are fully interconnected in the form of a directed graph, starting from the input to the output. All nodes except the input nodes have an associated activation function, which is used to compute the node output based on the weighted inputs from other nodes. An MLP model is trained through a back-propagation mechanism using gradient-descent as an optimization algorithm, where the weights between the nodes are adjusted iteratively for minimizing the error function.

3.1.4.1 Multilayer Perceptron - Classification

In classification, a relu activation function is used for all hidden layer nodes, and a softmax activation function is used for the output layer nodes. The output is a vector containing the probabilities that sample $x \in X$ belongs to each class, which is equivalent to a categorical probability distribution (as seen in Fig. 3.2). The final result is the class with the highest probability. With a categorical cross-entropy loss function, the network parameters are chosen to minimize the following:

$$E = -\sum_{l=1}^{C} b_{x,l} log(p_{x,l}) \tag{3.3}$$

where C is the number of classes, b is the binary indicator (0 or 1) whether class label l is the correct classification for input x, and p is the predicted probability that input x belongs to class l. Here, a separate loss is calculated for each class label per input, and the result is the sum of all those losses.

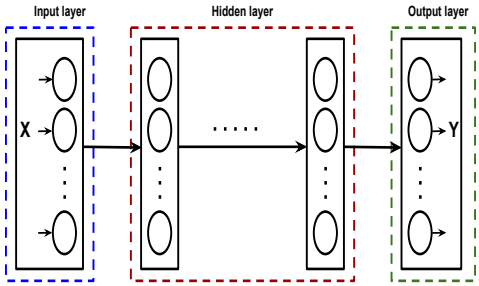


FIGURE 3.2: Structure of the MLP classifier model.

3.1.4.2 Multilayer Perceptron - Regression

In regression, a relu activation function is used for all hidden layer nodes, and a linear activation function is used for the output layer nodes. The output is a real-valued quantity predicted based on the input sample $x \in X$ (as seen in Fig. 3.3). With a Mean Squared Error (MSE) loss function, the network parameters are chosen to minimize the following:

$$MSE = 1/n \sum_{i=1}^{n} (Y_a - Y_p)^2 \tag{3.4}$$

where n is a vector of predictions generated from a sample of n data points while Y_a and Y_p are the actual and predicted values of the samples.

Hyperparameter Selection: Finding the parameters of a neural-network model means searching for the best hyper-parameters of the MLP that can

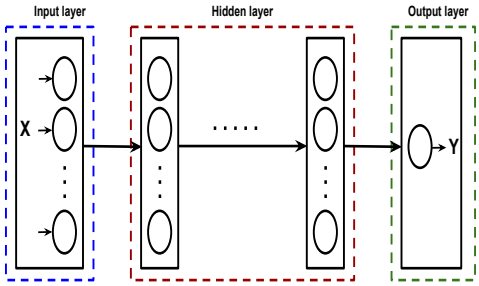


FIGURE 3.3: Structure of the MLP regressor model.

make the best predictions on the input. We applied *grid search* and *baby-sitting* as search strategies to perform an extensive search on the space of hyper-parameters to find the most accurate MLP classifier and regressor. This process included finding the number of hidden layers and nodes, the batch size, the regularization parameter, the learning rate of the optimizer, and the number of epochs. We encountered the process of finding hyper-parameters using grid search (or parameter sweep) method as time-consuming and hard because it is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm, which assures that this topic still requires significant research. Our search space for finding optimal hyperparameters for MLP models are as follows:

- Hidden layers: 1 to 5.
- Nodes in each hidden layer: 12 to 30 in intervals of 3.
- Optimizer: adam, SGD, RMSprop.
- Learning rate: 0.1, 0.01, 0.001.
- Batch size: 100 to 500 in intervals of 100.
- Number of epochs: 100 to 500 in intervals of 100.

We eventually found the architecture of the neural network that performs best on our traffic load traces and is described as follows. The structure includes one input layer with 12 nodes (i.e., one for each input feature), three hidden layers with $12, 24$ and 12 nodes, respectively, and an output layer with 10 nodes for MLP classifier (i.e., one for each output class) and 1 node for MLP regressor. The regularization parameter used is 0.01, the optimizer is based on stochastic gradient approach with a constant learning rate of 0.001, the batch size is fixed to 100, and the number of epochs equals 300.

3.1.5 Performance Evaluation

Keras is an open-source neural-network Python library capable of running on top of Theano [75] or TensorFlow [76]. It is characterized by a clean, uniform, and streamlined high-level Application Programming Interface (API), allowing users to rapidly define, train, and evaluate neural network models [77]. In Keras, the structure of the neural network model can be defined in a modular way, as a sequence of standalone and fully configurable modules, which can be readily plugged together. Keras offers several predefined neural layers such as a dense layer, a recurrent layer, and a convolutional layer. A wide range of activation functions is also available including relu, sigmoid, softmax, tanh, to name a few. Similarly, many predefined loss functions (e.g., MSE, cross entropy) and regularization schemes (e.g., dropout) are supported. Also, since Keras performs back-propagation automatically, users do not need to implement it. Moreover, numerous approaches are available to partition the dataset into training, validation, and test sets.

To implement an MLP in Keras, we construct a sequential model with a number of predefined dense layers and their corresponding activation functions. We then configure the learning process of the model by choosing an optimizer, a loss function (equation 3.3 or equation 3.4), and a list of metrics to be reported. Lastly, the model is trained with an objective to minimize the loss function and then evaluated.

We consider that ME Host nodes in proximity to the base stations are capable of hosting vUPF instances on their NFV infrastructure. We assume the link bandwidth capacity to be 20 Gbps and each vUPF can process a maximum of 200 Mbps traffic without QoS degradation. We consider horizontal vUPF auto-scaling with each ME host capable of hosting 100 $(20Gbps/200Mbps)$ vUPFs and $vUPF_{max} = 10$, i.e., a maximum of 10 vUPF can be hosted per cell. These assumptions are derived based on the evaluations performed by authors in [78]. If traffic load increases, additional vUPF instances are deployed to meet QoS/cost requirements, whereas if traffic load decreases, vUPF instances are removed to save operational expenses.

3.1.5.1 Multilayer Perceptron - Classification

Once the MLP classifier models are created as discussed before, a test dataset is used to assess the performance of the model in predicting outcomes. The

test outcomes can be classified into four groups: True Positive (TP) and True Negative (TN) are when the model correctly predicts actual positive and negative instances, respectively. Whereas, False Positive (FP) and False Negative (FN) are when the model makes incorrect predictions for negative and positive actual instances, respectively. Therefore, we consider four performance metrics to evaluate our MLP classifier model: accuracy, precision, recall, and f-measure, as given by equations 3.5, 3.6, 3.7 and 3.8, respectively.

$$Accuracy = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \tag{3.5}$$

$$Precision = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FP_i} \tag{3.6}$$

$$Recall = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FN_i} \tag{3.7}$$

$$F_{measure} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.8}$$

where C is the number of classes in the MLP model.

Accuracy is the most intuitive performance measure that gives the proportion of true predictions among the total number of predictions observed. However, accuracy is an excellent measure only if the datasets are entirely symmetric, i.e., FPs and FNs are almost the same. Therefore, other performance metrics need to be considered when evaluating a model. *Precision* is a measure of correctly predicted positive observations to the total predicted positive observations. It is a good measure to determine when the cost of FP is high. In the case of vUPF auto-scaling, a high number of FPs results in over-provisioning of resources leading to increased operational costs. On the other hand, *Recall* is a measure that calculates how many of the actual positives are captured in our model by labeling it as positive. It is a good measure to determine when the cost of FN is high. In the case of vUPF auto-scaling, a high number of FNs results in under-provisioning of resources leading to QoS degradation. Finally, *F-measure* is the weighted average of precision and recall, and it is used when there is an uneven class distribution.

Fig. 3.4 compares the performance of four proposed MLP classifier models: Q-classifier with scaling decisions every hour, Q-classifier with

scaling decisions every two hours, C-classifier with scaling decisions every hour, and C-classifier with scaling decisions every two hours. We use 6912 samples for training, 2304 samples for validation, and 2304 samples for testing. The Q-classifier/C-classifier with scaling decisions taken every hour outperforms other two models where scaling decisions are taken every two hours in all measures with 96.2% accuracy, 95.6% precision, 96% recall, and 96.2% f-measure.

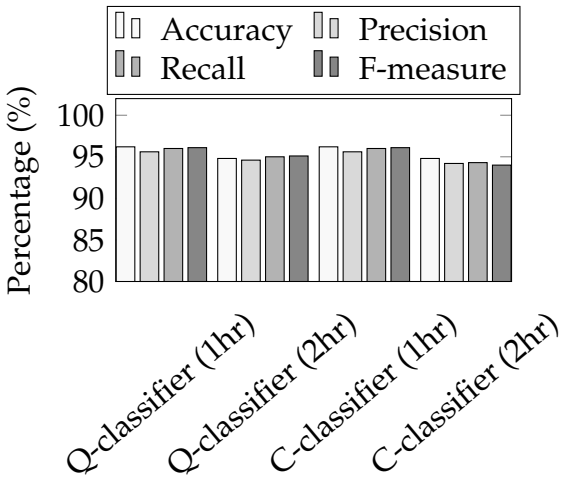


FIGURE 3.4: Comparision of the proposed MLP classifier models for vUPF auto-scaling.

Table 3.3, Table 3.4, and Table 3.5 reports the confusion matrix concerning the test data samples for Q-classifier/C-classifier models with scaling decisions every hour, Q-classifier model with scaling decisions every two hours, and C-classifier model with scaling decisions every two hours, respectively. It gives a breakdown of predictions into a table showing correct predictions (the diagonal) and the types of incorrect predictions made (what classes incorrect predictions were assigned). For example, if we observe Table 3.3 on the 2nd row, on 5 instances class 2 is misclassified as class 1, and on 6 instances class 2 is misclassified as class 3. Similarly, if we observe Table 3.4 on the 2nd row, on 2 instances class 2 is misclassified as class 1, and on 4 instances class 2 is misclassified as class 3, and on 1 instance class 2 is misclassified as class 4.

3.1.5.2 Multilayer Perceptron - Regression

Once the MLP regressor models are created as discussed before, a test dataset is used to assess the performance of the model in predicting outcomes. We implement four custom performance metrics in Keras to evaluate our MLP

Class	1	2	3	4	5	6	7	8	9	10
1	685	9	0	0	0	0	0	0	0	0
2	5	384	6	0	0	0	0	0	0	0
3	0	6	404	7	0	0	0	0	0	0
4	0	0	7	280	6	0	0	0	0	0
5	0	0	0	3	206	8	0	0	0	0
6	0	0	0	0	9	101	3	0	0	0
7	0	0	0	0	0	2	70	5	0	0
8	0	0	0	0	0	0	4	43	3	0
9	0	0	0	0	0	0	0	3	30	0
10	0	0	0	0	0	0	0	0	1	14

TABLE 3.3: Confusion matrix for the Q-classifier (1hr)/ C-classifier (1hr) models.

Class	1	2	3	4	5	6	7	8	9	10
1	322	6	0	0	0	0	0	0	0	0
2	2	165	4	1	0	0	0	0	0	0
3	0	3	206	5	0	0	0	0	0	0
4	0	0	2	142	6	0	0	0	0	0
5	0	0	0	2	112	6	1	0	0	0
6	0	0	0	0	6	53	1	0	0	0
7	0	0	0	0	0	3	41	4	0	0
8	0	0	0	0	0	0	3	24	1	0
9	0	0	0	0	0	0	0	3	19	0
10	0	0	0	0	0	0	0	0	0	9

TABLE 3.4: Confusion matrix for the Q-classifier (2hr) model.

Class	1	2	3	4	5	6	7	8	9	10
1	320	8	0	0	0	0	0	0	0	0
2	3	165	2	2	0	0	0	0	0	0
3	0	6	204	4	0	0	0	0	0	0
4	0	0	5	140	5	0	0	0	0	0
5	0	0	2	6	106	7	0	0	0	0
6	0	0	0	0	6	53	1	0	0	0
7	0	0	0	0	0	2	42	3	1	0
8	0	0	0	0	0	0	4	22	2	0
9	0	0	0	0	0	0	0	4	17	1
10	0	0	0	0	0	0	0	0	1	8

TABLE 3.5: Confusion matrix for the C-classifier (2hr) model.

regressor models: Mean Absolute Error (MAE), MSE, Root Mean Squared Error (RMSE), and R^2-score, as given by equations 3.9, 3.10, 3.11, and 3.12.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_a - Y_p| \tag{3.9}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_a - Y_p)^2 \tag{3.10}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_a - Y_p)^2} \tag{3.11}$$

$$R^2_{score} = 1 - \frac{\sum(Y_a - Y_p)^2}{\sum(Y_a - Y_m)^2} \tag{3.12}$$

where n is the number of test data samples, Y_a is the actual value of Y, Y_p is the predicted value of Y, and Y_m is the mean value of Y.

MAE estimates the average magnitude of errors in a set of forecasts, without considering their direction (i.e., the average of the absolute values of differences between the forecast and the corresponding observation).*MSE* measures the average of the squares of the errors (i.e., the average squared difference between the estimated values and the actual values). *RMSE* is a quadratic scoring rule which measures the average magnitude of the error (i.e., the difference between the estimated values and the actual values are each squared and then averaged over the sample). Then, the square root of the average is estimated. Considering the errors are squared before they are averaged, the RMSE adds a relatively high weight to big errors. Therefore, RMSE is most useful when large errors are undesirable. The RMSE is always larger or equal to the MAE, the greater difference between them, the higher the variance in the individual errors in the sample. If the RMSE is equal to the MAE, then all the errors are of the same magnitude. R^2*-score* (Coefficient of determination) represents the coefficient of how well the values fit compared to the original values. The value from 0 to 1 are interpreted as percentages. The higher the value is, the better the model is. It is equivalent to the accuracy metric in classification problems.

Fig. 3.5 compares the performance of four proposed MLP regressor models: Q-classifier with scaling decisions every hour, Q-classifier with scaling decisions every two hours, C-classifier with scaling decisions every hour, and C-classifier with scaling decisions every two hours. We use 6912 samples for training, 2304 samples for validation, and 2304 samples for testing. The Q-regressor/C-regressor (1hr) performs the best among the four models in all measures with 0.194 MAE, 0.0696 MSE, 0.194 RMSE and 98.43% R^2-score/accuracy. Moreover, considering that the MAE and the RMSE values are equal to each other, it is safe to say that the model has no large errors in the required number of vUPF predictions.

3.1.5.3 Classification vs Regression

Fig. 3.6 shows the prediction results of vUPF auto-scaling (for a full day) on test dataset for most reliable MLP classifier and MLP regressor models based on the metrics mentioned earlier, where we display the prediction performance on all six ME hosts, aggregated over all 10 cells for each base station. In the figure, the blue line represents the actual output generated from the dataset, the red line means the predicted vUPF scaling decisions

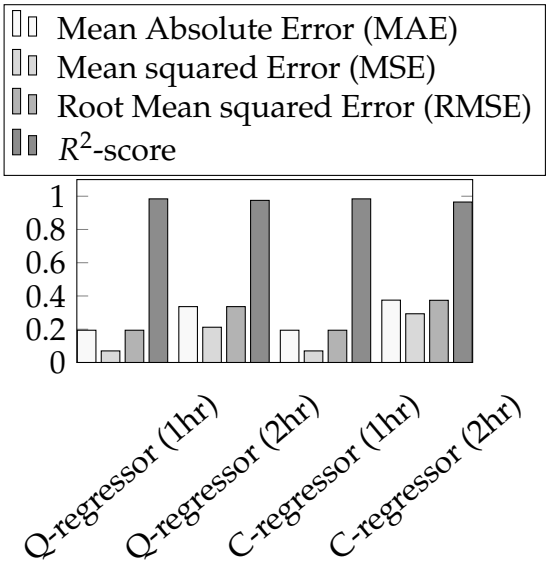


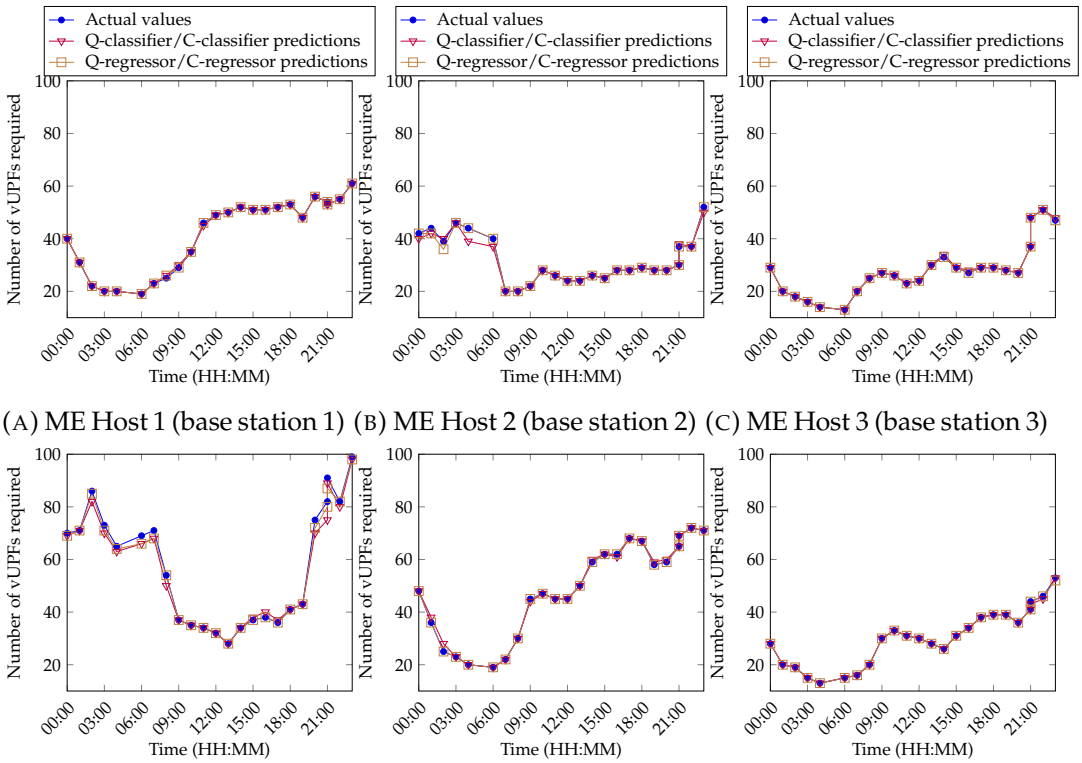
FIGURE 3.5: Comparision of the proposed MLP regressor models for vUPF auto-scaling.

using Q-classifier/C-classifier (1hr) models, and the brown line represents the predicted vUPF scaling decisions using Q-regressor/C-regressor (1hr) models. As we can observe, both classifier and regressor models introduced in this study can accurately follow the pattern of actual data, which point out the strong predicting capability of our models. However, Q-regressor/C-regressor (1hr) models (accuracy of 98.43% or R^2-score of 0.984) perform slightly better than Q-classifier/C-classifier (1hr) models (accuracy of 96.2%).

Fig. 3.7 depicts the MAE between the actual and predicted values of vUPF auto-scaling decisions on test dataset for best performing MLP classifier and MLP regressor models calculated over all six ME hosts for each hour during the entire day. In the figure, the red line represents the MAE for predicting vUPF scaling decisions using Q-classifier/C-classifier (1hr) models, and the brown line represents the MAE for predicting scaling decisions using Q-regressor/C-regressor (1hr) models. The former performs better than the latter with respect to MAE throughout the day.

3.1.6 Discussion

We proposed two ANN based MLP models (i.e., a classifier and a regressor) to facilitate proactive vUPF auto-scaling, based on the traffic traces obtained from a commercial operator. We evaluated the proposed models for its effectiveness in accurately predicting the amount of UPF instances required



(A) ME Host 1 (base station 1) (B) ME Host 2 (base station 2) (C) ME Host 3 (base station 3)

(D) ME Host 4 (base station 4) (E) ME Host 5 (base station 5) (F) ME Host 6 (base station 6)

FIGURE 3.6: Prediction results on the number of vUPFs required at each ME Host based on the proposed MLP models.

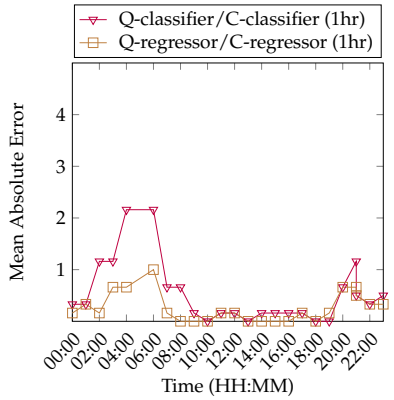


FIGURE 3.7: Comparison of Mean Absolute Error between the best performing MLP classifier and MLP regressor models.

as a function of the network traffic it should process. For MLP classifier, we measured accuracy, precision, recall, F-measure, and finally reported confusion matrix, while for MLP regressor we measured MSE, MAE, RMSE, and R^2-score. Our results show that both MLP classifier and MLP regressor models have strong predicting capability for auto-scaling. However, MLP regressor (1hr) models (accuracy of 98.43% or R^2-score of 0.984) outperforms MLP classifier (1hr) models (accuracy of 96.2%) in terms of accuracy.

It is worth mentioning that the predicted vUPF auto-scaling decisions in Q-regressor/C-regressor (1hr) MLP model (i.e. the best performing model) is used as input to evaluate the SFC placement model presented in Section 4.1. However, in doing so, we assume that UEs can be associated and served by any of the cells of a candidate base station, to simplify the problem.

3.2 Centralized Deep learning - Time Series Forecasting

3.2.1 Overview

This section considers a 5G mobile network composed of *three* base stations that are interconnected to each other through X_n interfaces and are served by their respective 5G core using NG interfaces, as shown in Figure. 3.8. Each base station in our network topology is equipped with a resource-constrained ME Host that is capable of hosting MEC applications. The rationale behind choosing such a 5G network topology is to align with the dataset that we obtained from the commercial network operator consisting of traffic load samples, for 43 consecutive days, only from three base stations. Assume that a generic MEC application is hosted on the ME Host as a container (i.e., application code and all its dependencies are packaged into a single image), which acts as a black-box entity performing specific tasks that require computation. Suppose each MEC application has a specific performance requirement on its target response time (D_{max}) that must be satisfied. Therefore, if the incoming workload on the MEC application increases, either the amount of computing resources assigned to the application needs to be increased or multiple application instances need to be instantiated. To this end, we exploit both horizontal and vertical auto-scaling of containerized MEC applications (i.e., Virtual MEC Application Functions (vMAFs)), which can introduce performance penalties (e.g., down-time), due to the enactment of the adaptation actions. The vMAFs applications can be instantiated quickly and reliably using container orchestration tools (e.g., Kubernetes), further simplifying the process of managing and orchestrating resources required to run those applications.

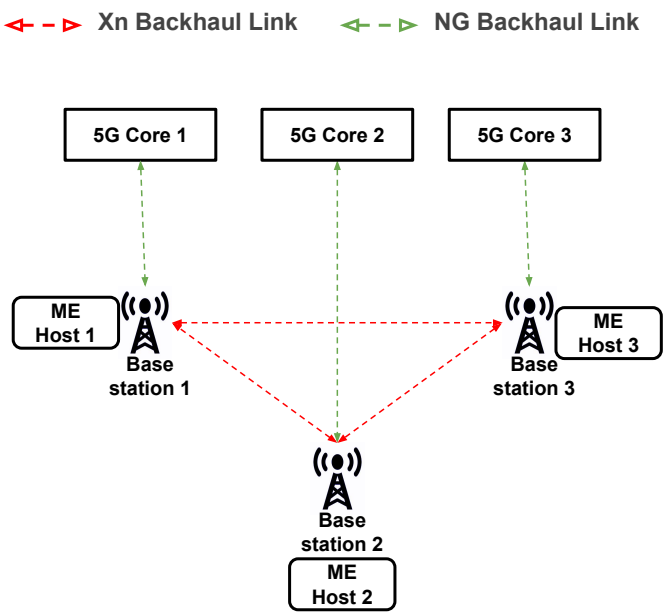


FIGURE 3.8: Single-domain (i.e., single operator) 5G Mobile Network.

3.2.2 Problem Statement

Given Dataset: The 5G mobile network operator dataset includes historical traffic load statistics X of individual mobile users, covering three base stations, for 43 consecutive days. The traffic load samples are in the form of time series $X = \{x[t_0], x[t_1], ..., x[t_{i-1}], x[t_i]\}$ and the traces are collected on an hourly timescale. The samples indicate the average traffic load per second for that hour and not the peak traffic load. So, the samples do not account for any short bursts (e.g., due to unexpected events) in traffic for that hour.

We observed that there exist a clear trend and seasonality (i.e., weekend spikes) in the traffic load samples for all three base stations. The randomness in the time series or the correlation among traffic load samples for all three base stations is computed, as shown in Fig. 3.9, through an auto-correlation plot. Due to the privacy constraints associated in open-sourcing our dataset, below we provide their statistical properties for each base station:

Base station 1: The traffic load samples are in the range of 0 - 5.85 GBPS with a mean of 0.89 GBPS, variance of 1.08 GBPS, skewness of 2.02 GBPS and kurtosis of 3.87 GBPS.

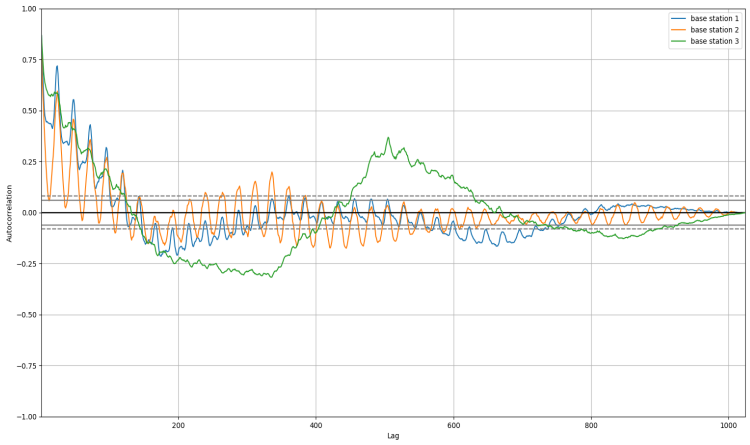


FIGURE 3.9: Auto-correlation Plot for all three Base Stations.

Base station 2: The traffic load samples are in the range of 0 - 4.01 GBPS with a mean of 0.45 GBPS, variance of 0.32 GBPS, skewness of 2.04 GBPS and kurtosis of 5.03 GBPS.

Base station 3: The traffic load samples are in the range of 0 - 4.79 GBPS with a mean of 0.60 GBPS, variance of 0.65 GBPS, skewness of 1.58 GBPS and kurtosis of 2.30 GBPS.

Find: Centralized neural network models that can predict the future values of $x[t_i]$ i.e., the actual number of vMAFs or CPU milllicores required at time $t_{i+1}, t_{i+2}, ..., t_{i+n}$. Then, compare the auto-scaling prediction performance of all neural network models developed.

Objective: Maximize QoS for the MEC application consumer or minimize operational cost for the service provider. The logic behind mapping the traffic load samples in X to the number of vMAFs or 100 millicore CPU units required to process that traffic load is based on the above-chosen objective. The accuracy of the prediction largely influences the achieved goal.

3.2.3 Data Transformation and Feature Engineering

In our traffic load dataset, extending backward from time t we have a time series of $\{x[t_i], x[t_{i-1}], ...\}$. We now want to estimate x at a future time t_{i+s}, where s is called the horizon of prediction. We can predict multiple time samples in the future. This is basically a function approximation problem which is represented by equation 3.13.

$$x[t_{i+s}] = f(x[t_i], x[t_{i-1}], ...) \tag{3.13}$$

We now transform this function approximation problem into a supervised learning problem by using previous time steps as input variables (X) and using the next time step as an output variable (y), and finally an algorithm is used to learn the mapping function from the input to the output. We propose two different approaches: (i) QoS prioritized time series forecasting (Y_Q) where the network operator gives priority to QoS over the cost; and (ii) Cost prioritized time series forecasting (Y_C) where the network operator chooses to neglect short-lived bursty traffic to avoid over-provisioning of vMAFs or 100 millicore CPU units. Therefore, the mapping of traffic load values (Y) to number of vMAFs or 100 millicore CPU units per vMAF is based on equation 3.14 and 3.15.

$$Y_Q(t) = min(vMAF_{max}, ceil(\frac{\lambda(t)}{\gamma})) \tag{3.14}$$

$$Y_C(t) = min(vMAF_{max}, floor(\frac{\lambda(t)}{\gamma})) \tag{3.15}$$

where $\lambda(t)$ is the traffic load in a base station at time t, γ is the maximum traffic load a single vMAF or 100 CPU millicore units can handle, and $vMAF_{max}$ is the maximum number of vMAFs that can be hosted on the MEC node or (maximum CPU millicore units)/100 that can be assigned to each vMAF.

The successive values of our time series can be preserved if we can set up a shift register of delays (e.g., like in LSTM), which allows each past value to be an additional spatial dimension in the input to our deep learning prediction model. Since the input space to the predictor needs to be finite, at each time instant t, we truncate the history to only previous z samples (z is called the embedding dimension).

Our dataset is decomposed into training, validation, and test datasets. We use a rule-of-thumb decomposition conforming to 60%/20%/20% between the training, validation, and test datasets, respectively. The data samples chosen for training and validation are the most balanced data in our dataset compared to samples that were used for testing. If the regression performance metrics indicate over-fitting, the K fold cross-validation technique can be used to generate multiple mini train-test splits to tune our neural-network models, which was not necessary in our case.

3.2.4 Proposed Centralized Deep Learning Models

Deep Neural Networks define parameterized functions from inputs to outputs as arrangements of many layers of fundamental building blocks called nonlinear functions. Common examples of nonlinear functions are sigmoids and Rectified Linear Units (ReLUs)). By adjusting the parameters of these nonlinear functions, such a parameterized function can be trained with the goal of fitting a given set of training samples. To be more specific, we define a loss function $L(\theta)$ over parameters θ, which is the average loss over all training samples $\{x_1, x_2, ..., x_n\}$. So, equation 3.16 represents the penalty for mismatch in the training data. The goal of training is to find the θ that results in the smallest loss, normally using the Stochastic Gradient Descent (SGD) technique. In SGD, a batch of samples is selected in random instead of the whole data set, for each iteration, and the gradient change is computed to update θ, based on the direction of gradient change, to achieve the local minimum.

$$L(\theta) = 1/N \sum_i L(\theta, x_i) \tag{3.16}$$

Time is continuously moving forward, and thus, it is difficult to deal with temporal data. Time series forecasting is challenging, mainly when dealing with long sequences, noisy data, multiple input/output variables, and multi-step forecasts. Deep learning techniques provide significant advantages for time series forecastings, such as automated learning of temporal dependence and automated handling of temporal structures such as trend and seasonality. In particular, LSTM neural network has been adopted mainly for time series forecasting. Adding convolutional layers to capture local, temporal patterns on top of LSTM layers can be immensely helpful in specific scenarios.

Hyperparameter Selection: Finding the parameters of a neural-network model means searching for the best hyperparameters (θ) that can make the best predictions on the input. We applied grid search and baby-sitting as search strategies to perform an extensive search on the space of hyperparameters to find the most accurate neural-network model. This process included finding the number of hidden layers and nodes, the batch size, the regularization parameter, the learning rate of the optimizer, and the number of epochs. Our search space for finding optimal hyperparameters for neural-network models are as follows:

- Hidden layers (h_n) - 1 to 10.

- Nodes per layer (d) - 24, 48, 72 and 96.
- Activation function per layer (A) - relu and tanh.
- Optimizers (O) - adam and SGD.
- Learning rates (lr) - 0.1, 0.01 and 0.001.
- Number of epochs (E) - 100 to 500 in intervals of 100.
- Loss functions (L) - huber and mean squared error.
- Batch size (B) - multiples of 48 upto 480.

3.2.4.1 Naive

A common naive forecasting technique is to use the persistence model, which estimates the value of the next step to be the same as that of the previous step. Nevertheless, this method performs very poorly on our seasonal data. Therefore, we consider a seasonal persistence model using the observed value from the previous day, i.e., $\hat{y}_{t+i|t} = y_{t+i-s}$, where s is the seasonal period (i.e., in our case s is 24 hours or 24 samples). We will use this seasonal persistence model as the benchmark for comparison with other time series forecasting approaches.

3.2.4.2 Artificial Neural Network (ANN)

Algorithm 1: The time series data, which includes one feature (i.e., number of vMAFs or number of 100 millicore CPU units per vMAF), is transformed into a supervised learning format with embedding dimension of z = 48, i.e., only 48 previous time steps are fed as input to the deep learning model. The data from the input layer is passed through 3 dense hidden layers with 48 or 24 hidden nodes per layer and using *relu* as the activation function in all nodes. The output layer has a single node for the regression output. For backpropagation, we use an efficient *adam* version of SGD with 0.01 learning rate, *huber* loss function, and 200 epochs. In each epoch, we calculate the loss (i.e., derivative) and then update the weight matrix for the next epoch based on the learning rate (lines 7-10). The objective is to minimize the loss on each epoch and to reach the local minimum. Once the model is defined and fit on the training data, the model can be used to predict for the next time step.

Algorithm 1 ANN for Time Series Forecasting

1: $h_1, h_2, h_3 \leftarrow$ dense layer
2: $Dropout\_Probablity \leftarrow 0.5$
3: $d_1, d_2 \leftarrow 48, d_3 \leftarrow 24$
4: $A_1, A_2, A_3 \leftarrow$ relu
5: $O \leftarrow$ adam, $L \leftarrow$ huber loss, $lr \leftarrow 0.01$, $E \leftarrow 200$
6: $train\_set \leftarrow base\_station\_1 + base\_station\_2 + base\_station\_3$
7: $model \leftarrow define(h_1, h_2, h_3, d_1, d_2, d_3, A_1, A_2, A_3)$
8: **for** $i = 1, i \leq E, i = i + 1$ **do**
9: $\quad L(i) \leftarrow compile(model, train\_set, metrics)$
10: $\quad new\_weights(i) \leftarrow weight\_update(L(i), O, lr)$
11: $\quad update(model, new\_weights(i))$
12: **end for**

3.2.4.3 Long Short-term Memory (LSTM)

Algorithm 2: The main difference in the LSTM model over the ANN model is that the data from the input layer is passed through 3 LSTM hidden layers with 48 hidden nodes per layer, so as to process each input sub-sequence of 48 time steps, and using *relu* as the activation function in all nodes. The output of the final LSTM hidden layer is followed by 2 dense hidden layers with 48 and 24 nodes, respectively, to interpret the summary of the input sequence. The backpropagation parameters are the same as that of the ANN model with an objective to minimize the prediction error by reaching the local minimum (lines 7-10).

Algorithm 2 LSTM for Time Series Forecasting

1: $h_1, h_2, h_3 \leftarrow$ lstm layer, $h_4, h_5 \leftarrow$ dense layer
2: $Dropout\_Probablity \leftarrow 0.5$
3: $d_1, d_2, d_3, d_4 \leftarrow 48, d_5 \leftarrow 24$
4: $A_1, A_2, A_3, A_4, A_5 \leftarrow$ relu
5: $O \leftarrow$ adam, $L \leftarrow$ huber loss, $lr \leftarrow 0.01$, $E \leftarrow 200$
6: $train\_set \leftarrow base\_station\_1 + base\_station\_2 + base\_station\_3$
7: $model \leftarrow define(h_1...h_5, d_1...d_5, A_1...A_5)$
8: **for** $i = 1, i \leq E, i = i + 1$ **do**
9: $\quad L(i) \leftarrow compile(model, train\_set, metrics)$
10: $\quad new\_weights(i) \leftarrow weight\_update(L(i), O, lr)$
11: $\quad update(model, new\_weights(i))$
12: **end for**

3.2.4.4 Convolutional Neural Network (CNN)

Algorithm 3: The main difference in the CNN-LSTM model over the LSTM model is that the data from the input layer is passed through 2 conv1D hidden layers, with 64 filters and a kernel size of 3, followed by a dropout layer for regularization. CNN layers can extract very informative, deep features, which are independent of time. CNN learn very quickly, so the dropout layer is needed to slow down the learning process, eventually resulting in a better final model. The output of the final conv1D hidden layer is followed by 3 LSTM hidden layers, each with 48 nodes, and 2 dense hidden layers with 48 and 24 nodes, respectively. All hidden layer nodes use *relu* as their activation function. The backpropagation parameters are the same as that of the ANN model with an objective to reach the local minimum (lines 7-10).

Algorithm 3 CNN-LSTM for Time Series Forecasting

1: $h_1, h_2 \leftarrow$ conv1D layer, $h_3, h_4, h_5 \leftarrow$ lstm layer, $h_6, h_7 \leftarrow$ dense layer
2: $Dropout\_Probablity \leftarrow 0.5$
3: $d_1, d_2, d_3, d_4, d_5, d_6 \leftarrow 48$, $d_7 \leftarrow 24$
4: $A_1, A_2, A_3, A_4, A_5, A_6, A_7 \leftarrow$ relu
5: $O \leftarrow$ adam, $L \leftarrow$ huber loss, $lr \leftarrow 0.01$, $E \leftarrow 200$
6: $train\_set \leftarrow base\_station\_1 + base\_station\_2 + base\_station\_3$
7: $model \leftarrow define(h_1...h_7, d_1...d_7, A_1...A_7)$
8: **for** $i = 1, i \leq E, i = i + 1$ **do**
9: $\quad L(i) \leftarrow compile(model, train\_set, metrics)$
10: $\quad new\_weights(i) \leftarrow weight\_update(L(i), O, lr)$
11: $\quad update(model, new\_weights(i))$
12: **end for**

3.2.4.5 Encoder-Decoder LSTM

Algorithm 4: The Encoder-Decoder LSTM model is comprised of two sub-models: the encoder and the decoder. The encoder is responsible for reading and interpreting the input sequence. The output of the encoder is a fixed-length vector that represents the model's interpretation of the time-series sequence. The output of the encoder is fed as input to the decoder.

The data from the input layer is fed to the LSTM hidden encoder layer, with 48 nodes, to read and encode the input sequences of 48 time steps. The encoded sequence is repeated three times by the model, for the 3 output time step predictions required, using a RepeatVector layer. This sequence is

then passed through 2 LSTM hidden decoder layers with 48 nodes per layer. Then, the sequence is passed through 2 dense hidden decoder layers with 48 and 24 nodes, respectively, before using a dense output layer wrapped in a TimeDistributed layer to produce one output for each time step in the output sequence. All nodes use *relu* as the activation function. For backpropagation, we use an efficient *adam* version of SGD with 0.01 learning rate, *mean_squared_error* loss function, and 500 epochs. In each epoch, we calculate the loss (i.e., derivative) and then update the weight matrix for the next epoch based on the learning rate so as to reach the local minimum (lines 9-11).

Algorithm 4 Encoder-Decoder LSTM for Multi-step Time Series Forecasting

1: $h_1, h_2, h_3 \leftarrow$ lstm layer, $h_4, h_5 \leftarrow$ dense layer
2: $r_1 \leftarrow$ repeatvector layer
3: $Dropout\_Probablity \leftarrow 0.5$
4: $d_1, d_2, d_3, d_4 \leftarrow 48, d_5 \leftarrow 24, d(r_1) \leftarrow 3$
5: $A_1, A_2, A_3, A_4, A_5 \leftarrow$ relu
6: $O \leftarrow$ adam, $L \leftarrow$ mse loss, $lr \leftarrow 0.01$, $E \leftarrow 200$
7: $train\_set \leftarrow base\_station\_1 + base\_station\_2 + base\_station\_3$
8: $model\_encoder \leftarrow define(h_1, r_1, d_1, d(r_1), A_1)$
9: $model\_decoder \leftarrow define(h_2...h_5, d_2...d_5, A_2...A_5)$
10: **for** $i = 1, i \leq E, i = i + 1$ **do**
11: $\quad L(i) \leftarrow compile(model\_encoder + model\_decoder, train\_set, metrics)$
12: $\quad new\_weights(i) \leftarrow weight\_update(L(i), O, lr)$
13: $\quad update(model\_encoder + model\_decoder, new\_weights(i))$
14: **end for**

3.2.4.6 Encoder-Decoder CNN-LSTM

Algorithm 5: The major difference in encoder-decoder CNN-LSTM model over encoder-decoder LSTM model is that data from the input layer is passed through 2 conv1D hidden encoder layers, with 64 filters and a kernel size of 3, followed by a max-pooling layer that summarizes the most activated presence of a feature. The encoded sequence is repeated three times by the model, for the 3 output time step predictions required, using a RepeatVector layer. This sequence is then passed through 3 LSTM hidden decoder layers and 2 dense hidden decoder layers, before using a dense output layer wrapped in a TimeDistributed layer to produce one output for each time step in the output sequence. All nodes use *relu* as the activation function. The backpropagation parameters are the same as that of the encoder-decoder LSTM model to reach the local minimum (lines 9-11).

Algorithm 5 Encoder-Decoder CNN-LSTM for Multi-step Time Series Forecasting

$h_1, h_2 \leftarrow$ conv1D layer, $h_3, h_4, h_5 \leftarrow$ lstm layer, $h_6, h_7 \leftarrow$ dense layer
$r_1 \leftarrow$ maxpooling layer, $r_2 \leftarrow$ repeatvector layer
$Dropout\_Probablity \leftarrow 0.5$
$d_1, d_2, d_3, d_4, d_5, d_6 \leftarrow 48$, $d_7 \leftarrow 24, d(r_1) \leftarrow 2, d(r_2) \leftarrow 3$
$A_1, A_2, A_3, A_4, A_5, A_6, A_7 \leftarrow$ relu
$O \leftarrow$ adam, $L \leftarrow$ mse loss, $lr \leftarrow 0.01$, $E \leftarrow 200$
$train\_set \leftarrow base\_station\_1 + base\_station\_2 + base\_station\_3$
$model\_encoder \leftarrow define(h_1, h_2, r_1, r_2, d_1, d_2, d(r_1), d(r_2), A_1, A_2)$
$model\_decoder \leftarrow define(h_3...h_7, d_3...d_7, A_3...A_7)$
for $i = 1, i \leq E, i = i + 1$ **do**
    $L(i) \leftarrow compile(model\_encoder + model\_decoder, train\_set, metrics)$
    $new\_weights(i) \leftarrow weight\_update(L(i), O, lr)$
    $update(model\_encoder + model\_decoder, new\_weights(i))$
end for

3.2.5 Performance Evaluation

TensorFlow [76] is an end-to-end open-source Python library for machine learning. It is characterized by a clean, uniform, and streamlined high-level API, allowing users to rapidly define, train, and evaluate machine learning models. In TensorFlow, the structure of the neural network model can be defined in a modular way using either the Sequential API or the Subclassing API, as a standalone and fully configurable module, which can be readily plugged together. TensorFlow offers several predefined neural layers, such as a dense layer, a recurrent layer, and a convolutional layer. A wide range of activation functions, predefined loss functions and regularization schemes are also supported.

We consider a mobile network scenario with three ME Hosts, as shown in Fig. 3.8 Suppose each of these MEC hosts is capable of hosting MEC applications or vMAFs on their NFV Infrastructure, as shown in Fig. 3. For horizontal vMAF auto-scaling, we assume that each ME host node can host a maximum of 10 vMAFs (this assumption is based on the fact that MEC hosts are resource-constrained as compared to the centralized Cloud, which can host 100's of vMAFs at a time, since they are located closer to the base station) where each of those vMAFs has 1000 millicore CPU units. For vertical auto-scaling, the scaling step size is set at 100 millicore CPU units (i.e., reaching up to a maximum of 1000 millicores per vMAF). As traffic load increases, additional vMAF/CPU instances are deployed to satisfy QoS requirements, whereas if traffic load declines, vMAF/CPU instances are

removed to minimize operational costs. Once the neural-network models are created as discussed earlier, the performance of these models is assessed based on correctly predicted outcomes in a test dataset. The centralized neural-network models are realized using the TensorFlow machine learning platform, while federated neural-network models are implemented using PyTorch and PySyft machine learning libraries.

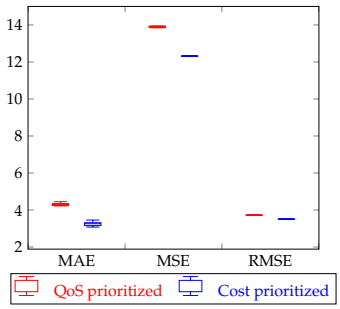
We evaluate our neural-network models using three performance metrics: MAE, MSE, and RMSE. MAE calculates the average magnitude of errors in a set of predictions, without recognizing their direction (i.e., the average of the absolute values of variations between the prediction and the corresponding observation). MSE estimates the average of the squares of the errors (i.e., the average squared deviation between the predicted values and the original values). RMSE is a quadratic scoring rule which estimates the average magnitude of the error (i.e., the difference between the estimated values and the original values are each squared and then averaged over the sample). Then, the square root of the average is calculated.

3.2.5.1 Naive vs ANN vs LSTM vs CNN-LSTM

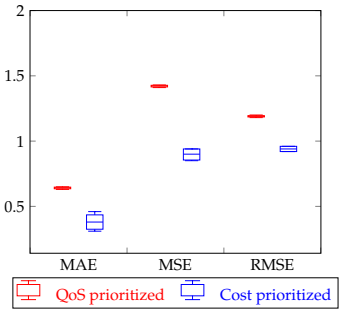
The box plots in Fig. 3.10 compares the performance of Naive, ANN, LSTM, and CNN-LSTM neural-network models for both QoS and cost prioritized auto-scaling objectives by performing 10 runs of simulation for each case and measuring MAE, MSE, and RMSE. Please note that we compare only the median values from the box plot. We use 3072 samples from the dataset and divide into training, validation, and test samples in the ratio of 60%:20%:20%.

For QoS prioritized auto-scaling, the CNN-LSTM neural-network model performs the best with 0.54 MAE, 1.36 MSE, and 1.16 RMSE. The second best is the LSTM model with 0.55 MAE, 1.39 MSE, and 1.17 RMSE. The third best is the ANN model with 0.64 MAE, 1.42 MSE, and 1.19 RMSE, and the worst performant is the Persistence model with 4.35 MAE, 13.94 MSE, and 3.73 RMSE.

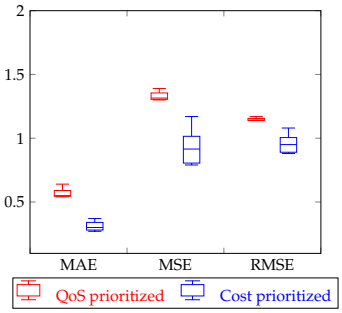
For cost prioritized auto-scaling, the LSTM neural-network model delivers the best at 0.31 MAE, 1.01 MSE, and 1 RMSE. The second best is the ANN model with 0.42 MAE, 0.94 MSE, and 0.96 RMSE. The third best is the CNN-LSTM model with 0.44 MAE, 1.44 MSE, and 1.2 RMSE, and the worst performant is the Persistence model with 3.28 MAE, 12.32 MSE, and 3.51 RMSE.



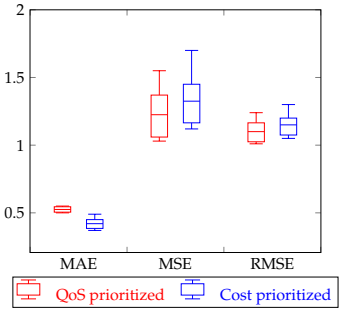
(A) Naive/Persistence model



(B) FFNN model



(C) LSTM model



(D) CNN-LSTM model

FIGURE 3.10: Comparision of the proposed neural-network models with QoS and Cost prioritized auto-scaling objectives.

3.2.5.2 (Encode-Decoder) LSTM vs CNN-LSTM

The box plots in Fig. 3.11 compares the performance of LSTM and CNN-LSTM encoder-decoder neural-network models with QoS prioritized auto-scaling objective by performing 10 simulation runs for each case and measuring MAE, MSE, and RMSE for 1st-step, 2nd-step, and 3rd-step predictions.

For 1st-step prediction, the CNN-LSTM encoder-decoder model measures with 0.06 MAE, 0.01 MSE, and 0.06 RMSE, and the LSTM encoder-decoder neural-network measures with 0.52 MAE, 0.58 MSE, and 0.52 RMSE.

For 2nd-step prediction, the CNN-LSTM encoder-decoder model measures with 0.08 MAE, 0.01 MSE, and 0.08 RMSE, and the LSTM encoder-decoder neural-network measures with 0.72 MAE, 1.15 MSE, and 0.72 RMSE.

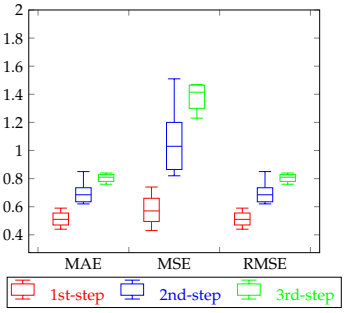
For 3rd-step prediction, the CNN-LSTM encoder-decoder model measures with 0.18 MAE, 0.07 MSE, and 0.18 RMSE, and the LSTM encoder-decoder neural-network measures with 0.82 MAE, 1.46 MSE, and 0.82 RMSE.

In all three cases, the CNN-LSTM model performs the best compared to the LSTM model. Also, the 1st-step predictions are better than the 2nd and 3rd-step predictions in both models.

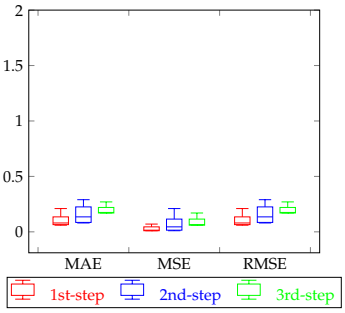
In summary, it is worth mentioning that based on the previous analysis from Fig. 3.10 and Fig. 3.11 we can conclude that although LSTM models constitute a popular and robust choice for load forecasting time series, their usage along with convolutional layers could provide a boost in the development of an efficient forecasting model. The reason being convolutional layers in CNN-LSTM will extract useful knowledge and learn the internal representation of time-series data while the LSTM layers identify the short-term and long-term dependencies.

3.2.6 Discussion

We design centralized DL techniques for predictive auto-scaling with QoS-prioritized and cost-prioritized objectives. We model the auto-scaling problem as a time series forecasting problem and determine one-step and multi-step future predictions using real-operator traffic load datasets for training, validation, and testing. We evaluate and compare Naive, ANN, LSTM, and CNN-LSTM models by measuring the MAE, MSE, and RMSE key performance metrics. For CL and one-step predictions, CNN-LSTM performs the best for the QoS-prioritized objective and LSTM performs the best for the cost-prioritized goal. For CL and multi-step predictions, the encoder-decoder CNN-LSTM model outperforms the encoder-decoder LSTM model.



(A) LSTM encoder-decoder model



(B) CNN-LSTM encoder-decoder model

FIGURE 3.11: Comparision of the proposed encoder-decoder neural-network models for 1st-step, 2nd-step and 3rd-step predictions.

3.3 Federated Deep Learning - Time Series Forecasting

3.3.1 Overview

This section considers a 5G mobile network composed of *three* base stations that are interconnected to each other through X_n interfaces and are served by their respective 5G core using *NG* interfaces. *Additionally, we consider that each base station and their corresponding 5G Core belongs to a unique operator (i.e., domain), as shown in Figure. 3.12.* Each base station in our network topology is equipped with a resource-constrained ME Host that is capable of hosting MEC applications. Similar to Section 3.2, we exploit both horizontal and vertical auto-scaling of containerized MEC applications (i.e., vMAFs), which can introduce performance penalties (e.g., down-time), due to the enactment of the adaptation actions. However, in contrast to the centralized model training techniques, FL allows ME hosts belonging to multiple domains to collaboratively learn a shared global model, without the need to transfer or share raw data across domains or to a centralized server during the training process. The objective of the training is to find the best

hyperparameters (θ) for the federated neural-network models that can make the best predictions on the input. Our search space for finding optimal hyperparameters is the same as that of centralized learning, which was discussed earlier. We adopt two federated learning approaches, i.e., with and without model averaging, to train the five neural-network models that were considered in the centralized approach (from Section 3.2.4).

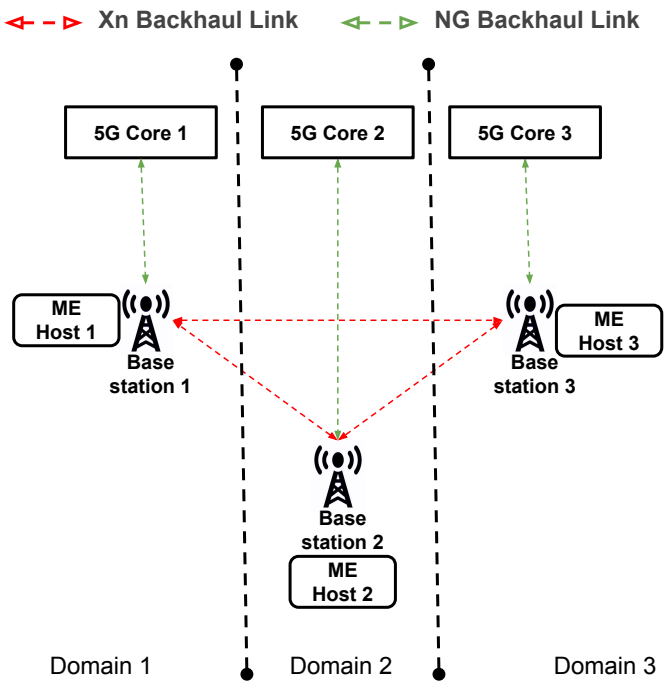


FIGURE 3.12: Multi-domain (i.e., multiple operators) 5G Mobile Network.

3.3.2 Problem Statement

Given Dataset: The dataset used is exactly the one used in Section 3.2.2. However, we assume that each base station belongs to different operator to demonstrate FL approach for vMAF auto-scaling.

Find: Federated neural network models that can predict the future values of $x[t_i]$ i.e., the actual number of vMAFs or CPU milllicores required at time $t_{i+1}, t_{i+2}, ..., t_{i+n}$. Then, compare the auto-scaling prediction performance of centralized and federated neural network models developed.

Objective: In addition to maximizing QoS for the MEC application consumer or minimizing operational cost for the service provider, to preserve the privacy of user data considering the multi-domain nature of

the considered 5G network. The logic behind mapping the traffic load samples in X to the number of vMAFs or 100 millicore CPU units required to process that traffic load is based on the above-chosen objective.

3.3.3 Data Transformation and Feature Engineering

The data transformation, feature engineering, and data decomposition for training, validation and testing of FL models are the same as that in Section 3.2.3. The only difference is that the data samples from three base stations are not combined but handled independently from one another as will be evident in the next subsection.

3.3.4 Proposed Federated Deep Learning Models

3.3.4.1 Without Model Averaging

Algorithm 6: Initially, we define a neural network model in the central server with the same architecture as previously used in CL. We call this model *global_model* (line 6). The training process includes three steps (i.e., since we have 3 virtual ME hosts) for each epoch. First, the central server sends the *global_model* and the training configuration to the 1*st* ME host. The 1*st* ME host now trains the model on its local training dataset and updates the weight matrix based on the calculated loss. The central server now receives the updated local model (with gradients) from the 1*st* ME host and assigns it to the *global_model* (lines 8-12). Now, the central server sends the updated *global_model* to the 2*nd* ME host. The 2*nd* ME host now trains the model on its local training dataset and updates the weight matrix based on the calculated loss. The central server now receives the updated local model (with gradients) from the 2*nd* ME host and assigns it to the *global_model* (lines 8-12). The same process occurs with the 3*rd* ME host (lines 8-12). For every epoch, the same procedure is repeated until the global loss function converges, or a desirable accuracy is reached (lines 7-13).

3.3.4.2 With Model Averaging

Algorithm 7: The previous approach of FL has some significant shortcomings to guarantee data privacy. Most notably, when the central server receives the updated model from the 1*st* ME host and sends the updated model to the 2*nd* ME host, the 2*nd* ME host can look at the gradients of 1*st* ME host and restore their training data to some extent. Therefore, we employ FL with a

Algorithm 6 Federated Deep Learning without Model Averaging

1: $h_1 ... h_7 \leftarrow$ conv1D or lstm or dense layer
2: $d_1 ... d_7 \leftarrow$ 48 or 24
3: $A_1 ... A_7 \leftarrow$ relu
4: $O \leftarrow$ SGD, $L \leftarrow$ huber loss, $lr \leftarrow 0.01$, $E \leftarrow 200$
5: $train\_set(1) \leftarrow base\_station(1), train\_set(2) \leftarrow base\_station(2), train\_set(3) \leftarrow base\_station(3)$
6: $global\_model \leftarrow define(h_1, ..., h_7, d_1, ..., d_7, A_1, ..., A_7)$
7: **for** $i = 1, i \leq E, i = i + 1$ **do**
8: **for** $j = 1, j \leq no.\_of\_ME\_hosts, j = j + 1$ **do**
9: $model(j) \leftarrow global\_model$
10: $L(j) \leftarrow compile(model(j), train\_set(j), metrics)$
11: $new\_weights(j) \leftarrow weight\_update(L(j), O, lr)$
12: $global\_model \leftarrow update(model(j), new\_weights(j))$
13: **end for**
14: **end for**

Model Averaging approach where the gradients from multiple ME hosts are averaged before updating the *global_model*. This algorithm has two parts to it: (i) training local models on each ME host and (ii) averaging the locally trained models on the central server before updating the *global_model*.

First, the central server sends the *global_model* and the training configuration to all three ME hosts. Each ME host now trains the model on its local training dataset and updates the weight matrix based on the calculated loss. The central server now receives the updated local models (with gradients) from all three ME hosts (lines 7-13). The central server aggregates the gradients received from all three ME hosts and updates its *global_model* (lines 14). In the next epoch, the updated *global_model* is now sent to all ME hosts, and the same procedure is repeated for every epoch until the global loss function converges, or a desirable accuracy is reached (lines 6-15).

3.3.5 Performance Evaluation

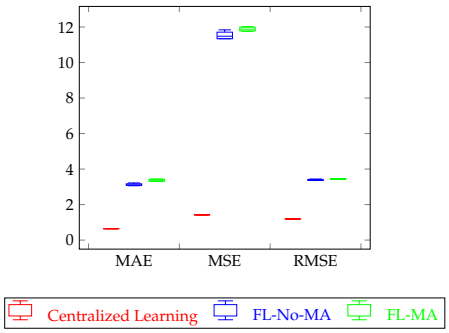
PyTorch is an open-source Python library that provides two high-level features: tensor computation (e.g., NumPy) and deep neural networks built on a tape-based autograd system. PyTorch also supports a rich ecosystem of tools and libraries to explore AI development. PySyft is one such Python library that allows us to handle remote tensors, remote models, and remote operations through virtual or physical worker objects for secure and private DL using FL. In our simulations, we create 4 virtual workers, i.e., 1 for central server (e.g., a trusted third party) and 3 for ME hosts.

Algorithm 7 Federated Learning with Model Averaging

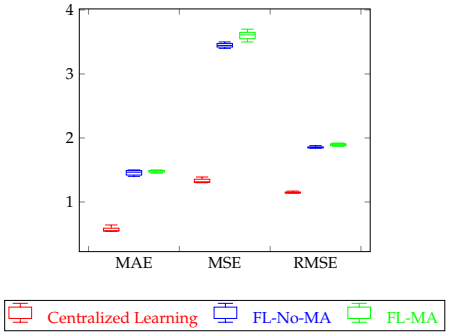
$h_1 ... h_7 \leftarrow$ conv1D or lstm or dense layer
$d_1 ... d_7 \leftarrow$ 48 or 24
$A_1 ... A_7 \leftarrow$ relu
$O \leftarrow$ SGD, $L \leftarrow$ huber loss, $lr \leftarrow 0.01$, $E \leftarrow 200$
$train\_set\_1 \leftarrow base\_station\_1, train\_set\_2 \leftarrow base\_station\_2, train\_set\_3 \leftarrow base\_station\_3$;
$global\_model \leftarrow define(h_1, ..., h_7, d_1, ..., d_7, A_1, ..., A_7)$
for $i = 1, i \leq E, i = i + 1$ **do**
    $model\_1, model\_2, model\_3 \leftarrow global\_model$
    $L_1(i) \leftarrow compile(model\_1, train\_set\_1, metrics)$
    $new\_weights_1(i) \leftarrow weight\_update(L_1(i), O, lr)$
    $L_2(i) \leftarrow compile(model\_2, train\_set\_2, metrics)$
    $new\_weights_2(i) \leftarrow weight\_update(L_2(i), O, lr)$
    $L_3(i) \leftarrow compile(model\_3, train\_set\_3, metrics)$
    $new\_weights_3(i) \leftarrow weight\_update(L_3(i), O, lr)$
    $global\_model \leftarrow federated\_avg(new\_weights_1(i), new\_weights_2(i), new\_weights_3(i))$
end for

3.3.5.1 Centralized Learning vs Federated Learning

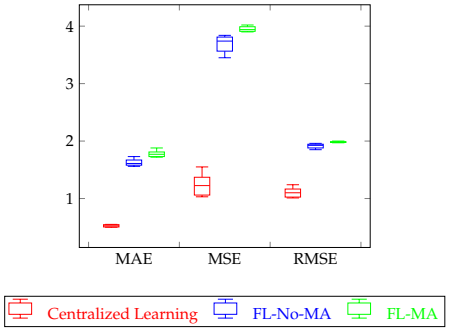
The box plots in Fig 3.13 compares the performance of CL models to FL models (i.e., with and without model averaging) trained on the distributed data (i.e., 3 virtual ME host nodes). The models are evaluated for QoS prioritized auto-scaling objective by running 10 simulations for each case and measuring MAE, MSE, and RMSE. The Table 3.6 also shows the comparison of MAE, MSE, and RMSE *median* values for centralized and federated learning neural network models. For each of the three neural network models (i.e., ANN, LSTM, and CNN-LSTM), the CL model performs the best. The second best is the FL without model averaging, and the worst performant is the FL with model averaging. The performance of FL models is poorer compared to the centralized approach, mainly because of the non-IID data distribution on each local ME host i.e., the skewness of data distribution [79]. The skewness can be represented as the distance between the data distribution on each local ME host and the population distribution. Such a distance can be evaluated with the Earth Mover's Distance (EMD) and is inversely proportional to the test accuracy.



(A) DNN model



(B) LSTM model



(C) CNN-LSTM model

FIGURE 3.13: Comparision of the proposed neural-network models with centralized, federated-without-ML and federated-ML algorithms.

3.3.6 Discussion

We design federated DL techniques for predictive auto-scaling with QoS-prioritized and cost-prioritized objectives. We model the auto-scaling problem as a time series forecasting problem and determine one-step future predictions using real-operator traffic load datasets for training,validation, and testing. We evaluate and compare ANN, LSTM, and CNN-LSTM CL and FL models by measuring the MAE, MSE, and RMSE key performance metrics. For FL, both LSTM and CNN-LSTM models perform equally better than the ANN model. Finally, FL performs poorly compared to centralized learning due to the non i.i.d. data samples in the individual local ME host nodes.

	FFNN			LSTM			CNN-LSTM		
Dataset	CL	FL - no MA	FL - MA	CL	FL - no MA	FL - MA	CL	FL - no MA	FL - MA
MAE	0.64	3.12	3.41	0.55	1.44	1.49	0.54	1.62	1.80
MSE	1.42	11.62	11.96	1.32	3.45	3.62	1.36	3.8	3.96
RMSE	1.19	3.40	3.45	1.14	1.85	1.9	1.16	1.94	1.98

TABLE 3.6: Comparison of MAE, MSE, and RMSE median values for centralized and federated neural network models.

Chapter 4

Service Function Chain Placement in 5G Networks

This chapter is based on the publication - [72]. In this chapter, we employ ILP technique to formulate and solve a 'joint UE association and SFC placement problem', where each SFC is composed of several VNFs interconnected through virtual links, with specific latency and data rate demands as requested by UEs positioned in diverse areas of the 5G mobile network. Furthermore, we also develop a comprehensive end-to-end latency model considering radio delay, backhaul network delay and SFC processing delay for 5G mobile networks. Subsequently, we also propose a heuristic algorithm with the same objective as that of ILP to address the scalability problem of ILP. Finally, we evaluate the performance of both ILP and heuristic solutions.

4.1 Overview

We consider a 5G mobile network, composed of *six* base stations, *two* aggregation points, and *one* 5G Core, as depicted in Fig. 4.1. A set of three base stations are interconnected to each other through *Xn*-interfaces. Using *NG*-interfaces, the six base stations are served by two aggregation points, and the 5G Core serves both of these aggregation points. It is to be noted that this network topology is exactly the same as the one we used in section 3.1. For simplicity, in the rest of this paper, we consider base station 1, base station 2, base station 3, and aggregation point 1 belong to cluster 1 while base station 4, base station 5, base station 6, and aggregation point 2 belong to cluster 2, as represented in Fig 4.1. Each element in our network topology is equipped with a resource-constrained (e.g., CPU) ME host that is capable of hosting SFCs composed of one or several VNFs (e.g., containers). We consider three feasible options for physically deploying ME hosts in 5G

networks, as defined by ETSI [80], i.e., ME host collocated with the base station, ME host collocated with aggregation point, and ME host collocated with 5G Core. Furthermore, in the considered hierarchical network topology, we assume that the closer is the ME host to the UE, the less is its computational capacity (e.g., ME host 1, ME host 2, and ME host 3 are identical nodes with least capacity, ME host 7 has the medium capacity, and ME host 9 has the highest capacity).

Suppose the UE (e.g., autonomous car) is associated with the base station 2 and requests for an SFC with an end-to-end latency (i.e., real-time, near real-time or non-real-time) and data rate requirements. The SFC requests considered in our work can be either of the three types as depicted in Fig. 4.1. ***The number of VNFs required is obtained via the horizontal auto-scaling prediction outputs from Section 3.1.*** Depending on the selected cost function to be minimized, the network provider can choose to place the VNFs of the SFC requested by the UE on either the host node (i.e., ME host 2) or any neighboring nodes (i.e., ME host 1, ME host 3) or distant nodes (ME host 7, ME host 9) or cluster 2 nodes (i.e., ME host 4, ME host 5, ME host 6, ME host 8) by allocating sufficient network resources (e.g., CPU, backhaul bandwidth), efficiently, while also making sure that the end-to-end latency and data rate requirements of the requested SFC is always satisfied. In the first case, no additional delay is introduced in the backhaul since the MEC node collocated with the host base station is the one hosting the VNFs. Conversely, in the other three cases, backhaul delay is introduced to map the virtual link onto a backhaul path, connecting the host base stations with a neighbouring ME host or a distant ME host or a ME host from a different cluster that is hosting VNFs.

4.2 Problem Statement

Given: a small 5G mobile network with base stations, aggregation points, 5G Core, ME hosts, the scheduling capabilities of base stations (e.g., PRBs, TTI duration, sub-carrier spacing), the computational capacity of each ME host, the transport network topology with the capacity of each backhaul link, the number of UEs and their requested SFCs with an end-to-end latency and data rate requirements.

Find: *'where'* to allocate resources to VNFs and *'which'* network paths to use.

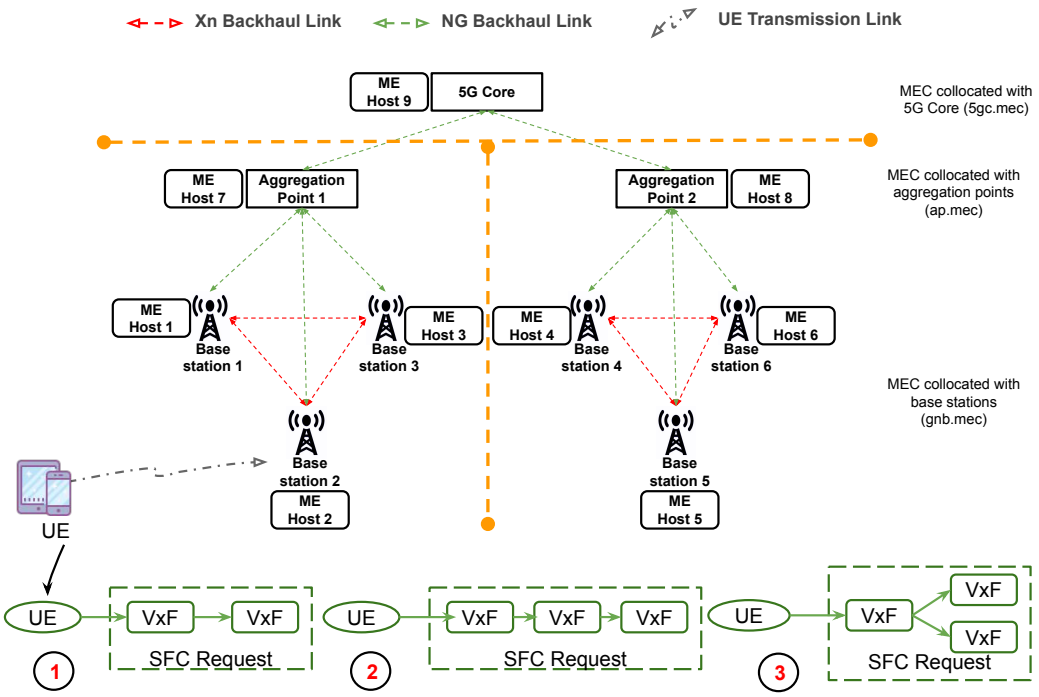


FIGURE 4.1: Substrate network topology and Service Function Chain requests

Objective: minimize average end-to-end latency for UEs to access their SFCs in the mobile network.

4.3 Proposed Models

4.3.1 5G Mobile Network Model

The mobile network infrastructure is modeled as an undirected graph $G_{net} = (N_{net}, E_{net})$, where $N_{net} = N_{gnb.mec} \cup N_{ap.mec} \cup N_{5gc.mec}$ is the union of the set of $|N_{gnb.mec}|$ base stations collocated with the ME host, $|N_{ap.mec}|$ aggregation points collocated with the ME host, and $|N_{5gc.mec}|$ 5G Core collocated with the ME host and E_{net} is the set of backhaul links such that an edge $e^{mn} \in E_{net}$ only if a connection exists between $m, n \in N_{net}$. Each network node $m \in N_{net}$ is attributed with a weight $w^{net}_{cpu}(m)$, representing its CPU capacity, under the assumption that one VNF requires one CPU unit to be instantiated. Additionally, each network node $m \in N_{gnb.mec}$ is also associated with a weight $w^{gnb.mec}_{prb}(m)$ representing the number of PRBs available at each timeslot that can be scheduled to UEs for transmitting data

packets. Furthermore, each edge $e^{mn} \in E_{net}$ is associated with a weight $w^{net}_{bw}(e^{mn})$ representing its bandwidth capacity (in Gbps). Finally, each network node $m \in N_{net}$ is associated with a geographical location $loc(m)$ (in terms of (x, y) coordinates) and each network node $m \in N_{gnb.mec}$ is associated with a coverage area $cov(m)$. Table 4.1 summarizes all the parameters used in the mobile network model.

Notation	Definition
G_{net}	Graph of the mobile network.
N_{net}	Set of all network nodes in G_{net}.
$N_{gnb.mec}$	Set of base stations collocated with the MEC node in G_{net}.
$N_{ap.mec}$	Set of aggregation points collocated with the MEC node in G_{net}.
$N_{5gc.mec}$	Set of 5GCs collocated with the MEC node in G_{net}.
E_{net}	Set of all backhaul links in G_{net}.
$w^{net}_{cpu}(m)$	Computing capacity of the network node $m \in N_{net}$.
$w^{gnb.mec}_{prb}(m)$	PRBs available for each timeslot at base station $m \in N_{gnb.mec}$.
$w^{net}_{bw}(e^{mn})$	Bandwidth capacity of the backhaul link $e^{mn} \in E_{net}$.
$loc(m)$	Geographical location of the network node $m \in N_{net}$.
$cov(m)$	Coverage area of the base station $m \in N_{gnb.mec}$.

TABLE 4.1: Parameters in the mobile network model.

4.3.2 Service Function Chain Request Model

Let $G_{req} = (N_{req}, E_{req})$ be a directed graph modeling the SFC requests, where $N_{req} = N_{ue} \cup N_{sfc}$ is the union of the set of $|N_{ue}|$ UEs and $|N_{sfc}|$ the set of SFCs requested from UEs and E_{req} is the set of virtual links between the UEs and their requested SFCs. Each SFC $s \in N_{sfc}$ is composed of a UPF (i.e., for encapsulation and decapsulation of GPRS Tunnelling Protocol for the user plane (GTP-U) of an UE requesting MEC services [80]) and one or more

vMAFs from a set of N_{vnfs}. Each SFC $s \in N_{sfc}$ is characterized by a maximum acceptable end-to-end latency (e.g., real-time, near real-time, non real-time) represented by $D_{E2E,max}(u,s)$ and a minimum guaranteed data rate denoted by $Thr_{req}(u,s)$ that needs to be satisfied. Each UE $u \in N_{ue}$ is associated with a location $loc(u)$ (in terms of (x,y) coordinates). Table 4.2 summarizes the parameters used in the SFC request model.

Notation	Definition
G_{req}	Graph of the SFC association request.
N_{req}	Set of all UEs and their SFC requests in G_{req}.
N_{ue}	Set of UEs in $Greq$.
$Nsfc$	Set of all the SFCs in G_{req}.
$Nvnfs$	Set of all the VNFs available to compose an SFC.
E_{req}	Set of all virtual links in G_{req}.
$D_{E2E,max}(u,s)$	Maximum acceptable end-to-end latency for a UE $u \in N_{ue}$ on its requested service $s \in N_{sfc}$.
$Thr_{req}(u,s)$	Requested data rate for a UE $u \in N_{ue}$ on the requested service $s \in N_{sfc}$.
$loc(u)$	Geographical location of the UE $u \in N_{ue}$.

TABLE 4.2: Parameters in the SFC request model.

4.3.3 UE Association, Scheduling Policy and Delay Model

In contrast to 4G technology where the goal is only to enhance the throughput of mobile broadband services, 5G is expected to support low-latency applications with end-to-end latency constraints of $1-10ms$ and error rates of 10^{-3} to 10^{-5} (e.g., connected cars). For cellular communications, two types of latency's are defined in 3GPP: control-plane (C-plane) latency and user-plane (U-plane) latency. The C-plane latency is the transition time for the UE to switch from idle mode to connected mode including the establishment of the user plane while U-plane latency is the one-way delay required to transmit a data packet from the UE to the mobile network (uplink) or vice-versa (downlink) [81].

In this work, we consider only the U-plane latency for computing end-to-end delay (D_{E2E}), since it is the major contributor that is hindering the support of URLLC applications. D_{E2E} is computed from the time UEs start transmitting packets in uplink until the time they start being processed in MEC nodes. For a scheduled UE, we assume we have 3 different communication delays contributing to D_{E2E}:

(i) ***Radio delay*** ($D_{radio}^{ue,gnb}$) is the sum of UE processing delay (t_{ue}^{proc}), over-the-air transmission delay (t_{TTI}), base station processing delay (t_{gnb}^{proc}), scheduler queuing delay (t_q), and Hybrid Automatic Repeat Request (HARQ) retransmission delay, which is given by equation 4.1,

$$\begin{aligned} D_{radio}^{ue,air,gnb} &= t_{ue}^{proc} + t_{TTI} + t_{gnb}^{proc} + t_q \\ &+2.n_{harq}(t_{ue}^{proc} + t_{TTI} + t_{gnb}^{proc} + t_q) \end{aligned} \tag{4.1}$$

where n_{harq} is the number of HARQ retransmissions required to achieve a Block Error (BLER) target of 10^{-3} to 10^{-5}. Similar to 3GPP, we adopt Orthogonal Frequency Division Multiplexing (OFDM) scheme. To satisfy the latency requirements of URLLC, 3GPP also proposes new frame structures with shorter TTI durations and multiple sub-carrier spacings. Scaling up the base subcarrier spacing of $15kHz$ by 2^{μ} (e.g., $30kHz, 60kHz$, and $120kHz$), the TTI duration of $1ms$ is scaled down by 2^{μ} (e.g., $0.5ms, 0.25ms$, and $0.125ms$), where $\mu = \{1, 2, .., n\}$, enabling faster transmission and lower processing time [82]. In our model, we adopt a TTI duration of $0.25ms$ resulting in a subcarrier spacing of $60kHz$ for all URLLC UEs. The resulting t_{ue}^{proc} and t_{gnb}^{proc} processing delays are 3 OFDM symbols and 1 TTI, respectively, as measured in [83]. The scheduler queuing delay (t_q), as represented in equation 4.2, is the sum of offset time (t_{offset}) i.e., the waiting time ($\sim$ 0 to 1 TTI) once the packet is ready for transmission until the beginning of the next TTI and the packet congestion time (t_{pktcon}) i.e., if the scheduler does not have enough PRBs to schedule a requested SFC packet in one TTI, the SFC packets may remain in the base station buffer for longer duration.

$$t_q = t_{offset} + t_{pktcon} \tag{4.2}$$

To determine t_{pktcon}, we first need to determine the number of PRBs ($N_{prb}(u, s, m)$) required for the SFC $s \in N_{sfc}$ of an UE $u \in N_{ue}$ to be assigned by its associated base station $m \in N_{gnb.mec}$. Given the data rate demand of the SFC s, the number of PRBs ($N_{prb}(u, s, m)$) is computed according to

equation 4.3 as given in 3GPP [84]:

$$N_{prb}(u,s,m) = \frac{Thr_{req}(u,s) * T_s^{\mu}}{12 * 10^{-6} * N_{cc} * N_{mimo} * N_{mod} * sf * R * (1-oh)} \quad (4.3)$$

where, N_{cc} is the number of aggregated component carriers, N_{mimo} is the number of Massive Input Massive Output (MIMO) layers, N_{mod} is the modulation order (e.g., 2 for Quadrature Phase Shift Keying (QPSK), 4 for 16Quadrature Amplitude Modulation (QAM), 6 for 64QAM, 8 for 256QAM), sf is the scaling factor, R is the code rate, oh is the overhead for control channels, and $T_s^{\mu} = 10^{-3}/(14 * 2^{\mu})$ is the average OFDM symbol duration in a subframe for numerology μ ($\mu = 2$ in our case) assuming normal cyclic prefix. Except for N_{mod} and R, which are determined as per the below three steps, all other parameters are predefined according to the radio access capabilities:

SINR measurement: The UE $u \in N_{ue}$ measures the Signal to Interference Noise Ratio (SINR) value for a reference signal coming from its associated base station $m \in N_{gnb.mec}$ using equation 4.4,

$$sinr(u,m) = \frac{\frac{F_m}{|loc(u)-loc(m)|^{\alpha}}}{\sum_{m' \neq m} \frac{F_{m'}}{|loc(u)-loc(m')|^{\alpha}} + N} \quad (4.4)$$

where $|loc(u) - loc(m)|$ is the distance between the UE u and its associated base station m, $|loc(u) - loc(m')|$ is the distance between the UE u and the neighbouring base stations of m (m'), α is a path loss exponent between 2 and 6, F_m and $F_{m'}$ are fading random variables of some distribution, and N is a constant noise term [85].

CQI report: The UE $u \in N_{ue}$ maps the SINR value measured in step 1 to a Channel Quality Indicator (CQI) index from the mapping table in [86], which is expected to be reported to the scheduler of its associated base station $m \in N_{gnb.mec}$. It is to be noted that these mappings are not defined in 3GPP but are vendor specific.

CQI to MCS mapping: The scheduler is now expected to map the reported CQI index to an Modulation and Coding Scheme (MCS) index and determine the best combination of modulation order (N_{mod}) and code rate (R) to be used from the mapping table in [87], resulting in a BLER target of 10^{-5}.

Therefore, we have all the necessary parameters in equation 4.3 to

determine the number of PRBs that must be assigned for an SFC $s \in N_{sfc}$ requested by the UE $u \in N_{ue}$ to meet its data rate requirements. If the number of PRBs that needs to be assigned are not available in a particular TTI, they will be assigned during the next TTI and so forth, adding up to the total t_q latency (i.e., t_{pktcon} = no. of TTIs to schedule SFC packets * TTI duration).

(ii) ***Backhaul delay*** ($D_{bh}^{Xn,NG}$) is the sum of X_n propagation delay (t_{Xn}^{prop}), X_n transmission delay (t_{Xn}^{tx}), NG propagation delay (t_{NG}^{prop}), and NG transmission delay (t_{NG}^{tx}), given by equation 4.5,

$$D_{bh}^{Xn,NG} = t_{Xn}^{prop} + t_{Xn}^{tx} + t_{NG}^{prop} + t_{NG}^{tx} \tag{4.5}$$

where for a link $e^{mn} \in E_{net}$, t_{Xn}^{prop} refers to the propagation time required to transmit SFC packets from node $m \in N_{gnb.mec}$ to node $n \in N_{gnb.mec}$ while t_{NG}^{prop} refers to the propagation time required to transmit SFC packets from node $m \in N_{gnb.mec}$ to node $n \in N_{ap.mec} | N_{5gc.mec}$. Similarly, t_{Xn}^{tx} and t_{NG}^{tx} refers to the transmission time required to transfer SFC packets from node $m \in N_{gnb.mec}$ and node $m \in N_{ap.mec} | N_{5gc.mec}$, to the outgoing link e^{mn}, respectively.

(iii) ***SFC processing delay*** (D_{mec}^{sfc}) is the time required for all VNFs in an SFC $s \in N_{sfc}$ to apply a specific network operation on the arriving packets.

Therefore, D_{E2E} is computed according to equation 4.6.

$$D_{E2E} = D_{radio}^{ue,air,gnb} + D_{bh}^{Xn,NG} + D_{mec}^{sfc} \tag{4.6}$$

It is to be noted that the same delay model can be used for both downlink and uplink direction.

4.4 Problem Formulation

Once, a batch of UE associations and its SFC requests arrive at the substrate network, it is either approved and embedded onto the network or it is denied. The embedding process includes both node and link mapping and is generally referred to as virtual network embedding problem which is proven to be NP-hard [88]. In the node mapping stage, each virtual node (i.e., UEs, VNFs in the SFCs requested by UEs) is mapped to a substrate node (i.e., base stations, ME hosts) while in the link mapping stage, each virtual

link (i.e., the link between the UE and its requested SFC) is mapped to a single substrate path (i.e. the path between the base station hosting the UE and ME hosts hosting the VNFs in the SFC). In both stages, the constraints imposed on substrate nodes and substrate links must be satisfied.

4.4.1 Integer Linear Programming

The proposed joint UE association and SFC placement problem is formulated employing ILP techniques. Before starting the actual problem formulation, for each UE $u \in N_{ue}$, we first determine the set of candidate base stations ($\overline{N}_{gnb.mec}(u)$) using equation 4.7,

$$\overline{N}_{gnb.mec}(u) = \{m \in N_{gnb.mec} | (|loc(u) - loc(m)|) \leq cov(m)\} \tag{4.7}$$

Then, we find neighboring base stations for each base station $m \in N_{gnb.mec}$ and neighboring ME nodes for each ME node $m \in N_{net}$ using equations 4.8 and 4.9, respectively.

$$nbr\_gnbs(m) = \{m' \in N_{gnb.mec} | e^{m,m'} \in E_{net}\} \tag{4.8}$$

$$\begin{aligned} nbr\_nodes(m) = \{&m' \in N_{gnb.mec}, m'' \in N_{ap.mec}, \\ &m''' \in N_{5gc.mec} | e^{m,m'}, e^{m,m''}, e^{m'',m'''} \in E_{net}\} \end{aligned} \tag{4.9}$$

Next, we find the candidate ME hosts that can host VNFs of SFC requested by UEs. For each VNF v of SFC s from the UE u, the set of candidate ME hosts ($\overline{N}_{net}(u, v, s)$) can be defined according to equation 4.10.

$$\begin{aligned} \overline{N}_{net}(u, v, s) = \{&m \in \overline{N}_{gnb.mec}(u), \\ &m' \in nbr\_gnbs(m), m'' \in N_{ap.mec}, \\ m''' \in N_{5gc.mec} | &e^{m,m'}, e^{m,m''}, e^{m'',m'''} \in E_{net}\} \end{aligned} \tag{4.10}$$

Thus, in our ILP model, either the UEs candidate base station ME host, or the base station ME host connected to the candidate base station ME host, or the aggregation point ME host connected to the candidate base station ME

host, or the 5G Core ME host connected to the aggregation point ME host serving the candidate base station ME host can host UEs SFC.

Now, we formulate the SFC placement problem with three binary decision variables, χ_m^u, $\Upsilon_m^{u,v,s}$, and $\Psi_{m,n}^{u,s}$, as represented in Table **??**.

Notation	Definition
χ_m^u	To show if $u \in N_{ue}$ is associated to base station $m \in N_{gnb.mec}$.
$\Upsilon_m^{u,v,s}$	To show if $v \in N_{vnfs}$ of $s \in N_{sfc}$ from $u \in N_{ue}$ is assigned to $m \in N_{net}$.
$\Psi_{m,n}^{u,s}$	To show if virtual link between $u \in N_{ue}$ and $s \in N_{sfc}$ is assigned to substrate link between $m \in N_{net}$ and $n \in nbr\_nodes(m)$.

The objective function of the ILP, given in equation 4.11, is to minimize the overall end-to-end latency from all users to their respective SFCs.

$$
\begin{aligned}
ILP : min[& \sum_{u \in N_{ue}} \sum_{m \in N_{gnb.mec}} \chi_m^u * D_{radio}^{ue,air,gnb}(u,m) \\
& + \sum_{u \in N_{ue}} \sum_{v \in N_{vnfs}} \sum_{s \in N_{sfc}} \sum_{m \in N_{net}} \Upsilon_m^{u,v,s} * D_{mec}^{sfc}(u,v,s,m) + \\
& \sum_{u \in N_{ue}} \sum_{s \in N_{sfc}} \sum_{m \in N_{net}} \sum_{n \in nbr\_nodes(m)} \Psi_{m,n}^{u,s} * D_{bh}^{Xn,NG}(u,s,m,n)] \quad (4.11)
\end{aligned}
$$

In equation 4.11, $D_{radio}^{ue,air,gnb}(u,m)$ depends on the number of UEs that need to be scheduled in a given time slot by base station m. For each UE u, we first find the $sinr(u,m)$ from equation 4.4 and then calculate the needed $N_{prb}(u,s,m)$ from equation 4.3 to transmit packets of SFC s with a particular size at a requested data rate ($Thr_{req}(s)$). If the total required PRBs exceed the maximum available PRBs in the base station, some UEs are scheduled in the next time slot, thus increasing the Radio delay for those UEs. Since the backhaul links, Xn and NG, in the mobile network, $D_{bh}^{Xn,NG}(u,s,m,n)$ depends on the the number of UEs sharing the same backhaul link.

We will now describe all node and link constraints imposed in our problem formulation. Constraint (4.12) ensures that each UE is associated to only one base station from its candidate set.

$$\sum_{m \in \overline{N}_{gnb.mec}(u)} \chi_m^u = 1, \forall u \in N_{ue} \tag{4.12}$$

Constraint (4.13) guarantees that each VNF of SFC requested from each UE is hosted by only one substrate ME host from its candidate set.

$$\sum_{m \in \overline{N}_{net}(u,s)} \Upsilon_m^{u,v,s} = 1, \forall u \in N_{ue} \forall s \in N_{sfc}^u \forall v \in N_{vnfs}^s \tag{4.13}$$

Constraint (4.14) guarantees that each VNF is at most shared by vnf_{max}^{shared} number of UEs.

$$\sum_{v \in N_{vnfs}} \Upsilon_m^{u,v,s} \leq vnf_{max}^{shared}, \forall u \in N_{ue} \forall s \in N_{sfc}^u \quad \forall m \in \overline{N}_{net}(u,s) \tag{4.14}$$

Constraint (4.15) ensures that the amount of CPU resources allocated to VNFs of SFCs adheres to the available CPU capabilities on the substrate node.

$$\sum_{u \in N_{ue}} \sum_{s \in N_{sfc}^u} \sum_{v \in N_{vnfs}^s} \Upsilon_m^{u,v,s} \leq w_{cpu}^{net}(m), \forall m \in N_{net} \tag{4.15}$$

Constraint (4.16) makes sure that in each time slot base stations can associate UEs only if they have enough PRBs to meet the data rate demand of the requested SFC by the UE.

$$\sum_{u \in N_{ue}} \sum_{s \in N_{sfc}} N_{prb}(u,s,m) * \chi_m^u \leq w_{prb}^{gnb.mec}(m), \quad \forall m \in N_{gnb.mec} \tag{4.16}$$

Flow constraint (4.17) enforces for each virtual link between UE $u \in N_{ue}$ and its SFC $s \in N_{sfc}$ there exists a continuous path established between the base station to which the UE is associated and the ME host hosting the VNFs of SFC s.

$$\sum_{n \in nbr\_nodes(m)} (\Psi_{n,m}^{u,s} - \Psi_{m,n}^{u,s}) = \Upsilon_m^{u,s} - \chi_m^u, \quad \forall m \in N_{net}, \forall e^{u,s} \in E_{req} \tag{4.17}$$

Constraint (4.18) makes sure that virtual links are mapped onto the backhaul substrate links in the mobile network, if and only if it has enough bandwidth capacity to meet the link demand of virtual links.

$$\sum_{u \in N_{ue}} \sum_{s \in N_{sfc}} Thr_{req}(u,s)(\Psi^{u,s}_{n,m} + \Psi^{u,s}_{m,n}) \leq w^{net}_{bw}(e^{nm}),$$
$$\forall m \in N_{net}, \forall n \in nbr\_nodes(m), n < m \quad (4.18)$$

Constraint (4.19) ensures that the end-to-end latency from the UEs to its associated SFCs does not exceed the maximum acceptable latency as requested by the UEs.

$$\sum_{m \in N_{gnb.mec}} \chi^{u}_{m} * D^{ue,air,gnb}_{radio}(u,m) + \sum_{m \in N_{net}} \sum_{v \in N^{s}_{vnfs}} \Upsilon^{u,v,s}_{m} * D^{sfc}_{mec}(u,v,s,m) + \sum_{m \in N_{net}} \sum_{n \in nbr\_nodes(m)} \Psi^{u,s}_{m,n} * D^{Xn,NG}_{bh}(u,s,m,n) \leq D_{E2E,max}(u,s), \forall u \in N_{ue}, \forall s \in N^{u}_{sfc} \quad (4.19)$$

4.4.2 Heuristic

The above ILP formulation took 44 hours to associate 300 UEs including their latency-sensitive SFC requests composed of a number of VNFs on a mobile network comprised of *six* base stations, *two* aggregation points, and *one* 5G core. The ILP was solved using ILOG CPLEX solver on an Intel Core i7 laptop with 3GHz CPU and 16 GB RAM. To address the issue of scalability in ILP, we propose a heuristic algorithm, as seen in Algorithm 8, that can solve the above association/mapping problem in a couple of seconds. Similar to the ILP-based algorithm, the objective of our heuristic algorithm is to minimize the overall end-to-end latency from all UEs to their requested SFCs.

In the first step (lines $1-10$), the algorithm loops through all the UEs to determine a set of candidate base stations considering the location of the UE, location of the base station, and the coverage area of the base station, and then creates a list of $cand\_gnb(u)$ for each UE. Next, each UE is mapped to the base station, among the $cand\_gnb(u)$, that measures the best signal quality (i.e., SINR) and also has sufficient PRBs to host the UE.

In the second step (lines $11-20$), the algorithm finds the candidate ME hosts for each VNF of SFC requests received from all UEs, which is nothing but the union of the ME host collocated with the host base station of UE (determined from step 1) and all other ME hosts connected directly or indirectly to the host base station of UE through backhaul links. Another list of $cand\_mec(u, s, v)$ is created for each VNF of the SFC received from all UEs.

In the third step (lines $21-38$), the algorithm begins mapping all VNFs of SFC requests with real-time latency requirements considering $cand\_mec(u, s)$ for each SFC, starting from $gnb.mec$ nodes. Once they run out of computing resources, the algorithm moves on to $ap.mec$ nodes, and finally on to $5gc.mec$ nodes, if and only if the computed end-to-end latency ($D_{E2E}(u, s, m)$) is less than the maximum acceptable end-to-end latency for that SFC ($D_{E2E,max}$). Moreover, if an instance of VNF is already mapped to the candidate ME host the UE shares this VNF to realize its SFC instead of instantiating a new VNF. The heuristic then uses the shortest path algorithm to map the virtual link between the UE and its requested SFC onto the substrate link between the base station that the UE is associated to and the ME host that the SFC is being hosted on. The VNFs of a single SFC might be mapped on different ME hosts, and therefore further caution is exercised during link mapping. The node and link computational resources are updated after each mapping. The same process is repeated for other SFC requests with near-real-time and non-real-time latency requirements until all SFC requests are mapped.

4.5 Performance Evaluation

The performance of the latency-optimal SFC placement ILP model is evaluated based on the simulations implemented in Python. We then compare it to the implemented heuristic algorithms performance. Real-operator network topology and realistic latency values are used when modeling the simulation environment to produce realistic simulation results, which can better illustrate the benefits of placing SFCs composed of VNFs at the network edges closer to the end-user.

4.5.1 Simulation Environment

A small cluster of 5G mobile network composed of 9 network nodes is considered in our simulation, as depicted in Fig. 4.1. A set of 3 base stations are connected to each other through 20 Gbps Xn backhaul links, while each

Algorithm 8 Heuristic

Require: G_{net}, G_{req}, and SFC latency budget [$N_{sfc}(rt)$, $N_{sfc}(near\_rt)$, $N_{sfc}(non\_rt)$].
Ensure: User association and latency-optimal SFC placement.

Step 1. *Find candidate gNodeBs for each UE and perform UE association.*

for u in N_{ue} **do**
  $cand\_gnb(u) \leftarrow 0$
  $map\_gnb(u) \leftarrow 0$
  for m in $N_{gnb.mec}$ **do**
    if $|loc(u) - loc(m)| <= cov(m)$ **then**
      $cand\_gnb(u) \leftarrow m$
    end if
  end for
  $map\_gnb(u) \leftarrow m$ from the list of $cand\_gnb(u)$ with $max(sinr(u,m))$ and enough PRBs available.
end for

Step 2. *Find candidate MEC nodes for VNFs of each SFC from each UE.*

for u in N_{ue} **do**
  for s in N_{sfc} **do**
    for v in $N_{sfc}(u)$ **do**
      $cand\_mec(u,s,v) \leftarrow 0$
      for m in $neighbours(map\_gnb(u))$ **do**
        $cand\_mec(u,s,v) \leftarrow m$
      end for
    end for
  end for
end for

Step 3. *Perform SFC placement for each UE.*

for u in N_{ue} **do**
  for s in $N_{sfc}(rt)$ **do** /* *real-time SFCs.* */
    for v in $N_{sfc}(u)$ **do**
      $map\_mec(u,s,v) \leftarrow 0$
      for m in $cand\_mec(u,s,v)$ **do**
        $compute(D_{E2E}(u,s,m))$
        if $D_{E2E}(u,s,m) <= D_{E2E,max}$ **then**
          if $inst(v)$ not in m or $neighbours(m)$ **then**
            $map\_mec(u,s,v) \leftarrow m$
          end if
        end if
        $alocate\_continuous\_path(u,s,m)$
        $update\_node\_and\_link\_resources()$
      end for
    end for
  end for
end for

Repeat Step 3 *for s in* $N_{sfc}(near\_rt)$ *and* $N_{sfc}(non\_rt)$.

of the three base stations is connected to the aggregation point using 20 Gbps *NG* backhaul links which in turn is connected to the 5G Core using 50

Gbps backhaul links. The number of aggregated component carriers is set to 4, and each carrier has a bandwidth capacity of 20 MHz. We assume that the base stations support $4x4$ MIMO configuration. We then introduce ME hosts at each of these 9 network nodes capable of hosting a limited number of VNFs. The ME hosts collocated with base stations each have 50 CPUs, the ME hosts collocated with aggregation points each have 100 CPUs, and the ME hosts collocated with 5G Core has 500 CPUs.

Our simulations are carried out for two scenarios. In the first scenario, we consider that SFC requests arrive in batches of 30 UEs (equally divided among real-time, near-real-time, and non-real-time) with each batch corresponding to 1 timeslot. In every timeslot, the ILP considers the SFC requests received in previous batches and associates all the UEs and their SFC requests onto the mobile network, considering the latency and data rate requirements of each SFC. We consider 10 batches of SFC requests corresponding to 300 UEs. In the second scenario, we consider that SFC requests arrive according to the *predicted number of vUPF instances from our MLP neural-network model, as illustrated in Section 3.1*. The performance of ILP is compared with our heuristic algorithm in both scenarios. Additionally, in both scenarios, we assume that each UPF instance corresponds to one UE and each SFC is composed of 2 or 3 VNFs (1 vUPF and 1 or 2 vMAF as depicted in Fig. 4.1) with 1 CPU required to instantiate every VNF. Furthermore, each vUPF is shared with 5 UEs while each vMAF is shared among 2 UEs. Since the UEs considered in our model are URLLC UEs, we assume three categories of user-to-SFC one-way delay requirements, i.e., $1ms$, $2ms$, and $5ms$. We assume that each UE is transmitting short packets of size $15Kb$ every TTI and requests a minimum data rate of $200Mbps$. In Section 4.3, we discussed on how we calculate $D_{Radio}^{ue,air,gnb}$. We calculate $D_{bh}^{Xn,NG}$ by dividing the total packet size generated from all the UEs that are using the same backhaul link with the bandwidth capacity of the link [89]. Finally, we calculate D_{mec}^{sfc} by dividing the packet size that the VNFs of the SFC should process by the CPU speed. We consider a CPU speed of $3GHz$ with 64 bit processor.

4.5.2 Simulation Results

CPU Utilization: The CPU utilization is computed by dividing the number of CPUs utilized in $gnb.mec$ nodes or $ap.mec$ nodes or $5gc.mec$ nodes once the VNFs are mapped to the total number of CPUs available in all $gnb.mec$ nodes or all $ap.mec$ nodes or all $5gc.mec$ nodes, respectively.

Fig. 4.2 illustrates the CPU utilization of ME hosts with respect to the number of UEs for simulations carried out in scenario 1. We observe that up to ≈ 90 UEs, the ILP places most of the VNFs on *gnb.mec* nodes because of its proximity to UEs, irrespective of the SFC latency requirements, while some non-real-time VNFs that are shared by UEs associated with cluster 1 (base station 1, base station 2 or base station 3) and cluster 2 (base station 4, base station 5 or base station 6) are placed on 5*gc.mec* nodes. Only after *gnb.mec* nodes are depleted with their CPU resources (after 90 UEs), the ILP starts moving VNFs with near-real-time and non-real-time latency requirements initially placed in *gnb.mec* nodes to *ap.mec* nodes and starts placing new SFCs with real-time latency requirements on *gnb.mec* nodes. Similarly, when CPU resources of *ap.mec* nodes are depleted (≈ 180 UEs) the ILP starts moving VNFs with non-real-time latency requirements initially placed in *gnb.mec* or *ap.mec* nodes to 5*gc.mec* nodes. On the other hand, heuristic algorithm follows a similar pattern to that of ILP, but instead of placing non-real-time VNFs that are shared by UEs associated to cluster 1 and cluster 2 on 5*gc.mec* nodes, those VNFs are initially placed on *gnb.mec* nodes, then on *ap.mec* nodes and finally on 5*gc.mec* nodes (in this order depending on the ME hosts resource availability). This is evident from Fig. 4.2, where CPU utilization of *gnb.mec* nodes is always higher in heuristic compared to that of ILP. However, this results in the increase of overall latency for heuristic due to the users taking a long path to access their SFC services (e.g., if a user is associated to base station 2 and its VNFs are placed in *gNB6.mec*, the path mapping could be base station 2 → aggregation point 1 → 5G Core → aggregation point 2 → base station6). Consequently, up to 180 UEs, the CPU utilization of *ap.mec* and 5*gc.mec* nodes are lower in heuristic compared to ILP.

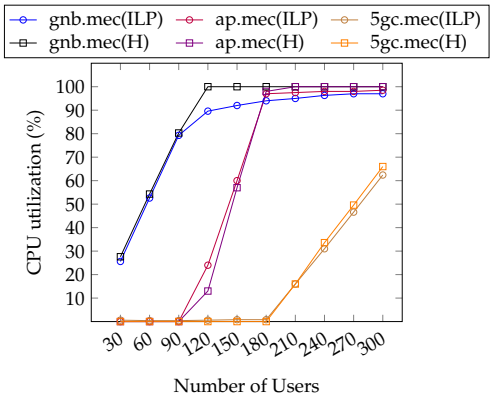


FIGURE 4.2: CPU utilization of MEC nodes (Scenario 1).

Fig. 4.3 illustrates the CPU utilization of ME hosts with respect to time over one full day for the network considered in the MLP neural-network model (scenario 2). We observe that the CPU utilization of *gnb.mec* nodes are always full, the *ap.mec* nodes are most of the time full except from 2:00 to 8:00 and 5*gc.mec* nodes have low utilization during early morning (2:00 to 8:00) due to the low number of UEs being active and the utilization gradually increases during the day peaking late in the night ($\approx$ 22:00). The heuristic follows a similar pattern to that of ILP, but as discussed earlier VNFs shared by UEs belonging to different clusters are initially placed on *gnb.mec* nodes rather than on 5*gc.mec* nodes like in ILP. Therefore, CPU utilization for *gnb.mec* nodes are always higher in heuristic compared to that of ILP, with a tradeoff being the increase in overall latency.

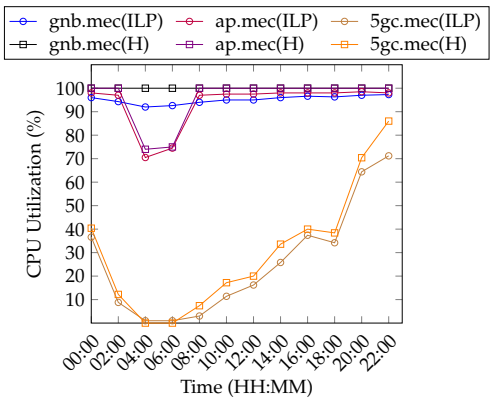


FIGURE 4.3: CPU utilization of MEC nodes (Scenario 2).

Link utilization: Link utilization is calculated by dividing the usage of either *Xn* or *NG* backhaul links by UEs for utilizing SFCs in the ME hosts to the total available capacity of the respective links.

Fig. 4.4 and Fig. 4.5 illustrates, respectively, the *Xn* link utilization and the *NG* link utilization as a function of the number of UEs for experiments carried out in scenario 1. In Fig. 4.4, we observe that in ILP, irrespective of the number of UEs, the *Xn* link utilization remains almost the same ($< 3\%$), which is attributed to the fact that ILP principally places the VNFs of UEs on the *gnb.mec* that is currently serving the corresponding UE over the air interface in order to minimize the end-to-end latency. However, we observe that heuristic algorithm places VNFs of some UEs on *gnb.mec* nodes that are currently not serving the corresponding UE over the air interface, which leads to the usage of *Xn* links. After a certain point ($\approx$ 90 UEs in Fig. 4.4), the *Xn* link utilization remains almost the same for heuristic since the capacity of all *gnb.mec* nodes are depleted, and VNFs are placed on *ap.mec* or 5*gc.mec*

nodes there on. In Fig. 4.5, we can observe that both in ILP and heuristic *NG* links are least utilized up to ≈ 90 UEs since most VNFs of SFCs, irrespective of their latency demands, are always placed on *gnode.mec* nodes until then. Once *gnode.mec* nodes are out of CPU resources, the VNFs of SFCs are moved to *ap.mec* nodes and later to 5*gc.mec* nodes considering the latency requirements of SFCs, resulting in the significant usage of *NG* backhaul links. However, the reason for higher *NG* link utilization in heuristic is attributed to the fact that some UEs take longer routes, from cluster 1 to cluster 2 or vice-versa, in order to access their SFC which is not the case in ILP.

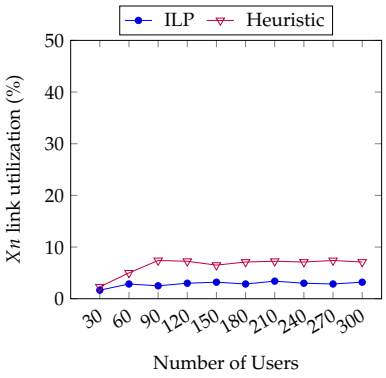


FIGURE 4.4: Xn-link (base station-to-base station) utilization (Scenario 1).

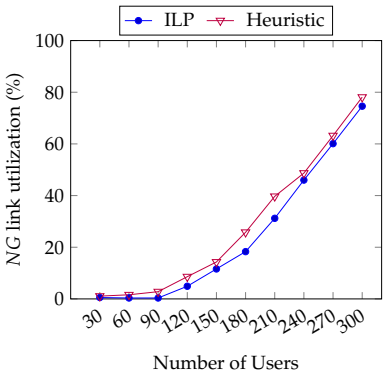


FIGURE 4.5: NG-link (base station-to-AP-to-5GC) utilization (Scenario 1).

Fig. 4.6 and Fig. 4.7 illustrates, respectively, the *Xn* link utilization and the *NG* link utilization with respect to time over one full day based on the network considered in the MLP neural-network model (Scenario 2). Since the number of UEs is always more than 90, we observe that both ILP and heuristic algorithm places VNFs of some UEs on *gnb.mec* nodes that are currently not serving the corresponding UE over the air interface which leads to the usage

of Xn links as already explained in Scenario 1. Likewise, NG link utilization for both ILP and heuristic is lowest during early morning (2:00 to 8:00) due to the low number of UEs being active and the utilization gradually increases during the day peaking late in the night ($\approx$ 22:00). However, both Xn and NG link utilizations in heuristic are higher compared to ILP because of the long path the UEs take to access SFC like we discussed before.

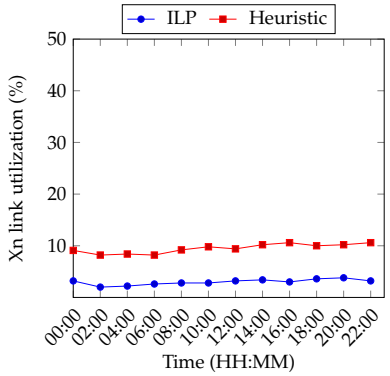


FIGURE 4.6: Xn-link (base station-to-base station) utilization (Scenario 2).

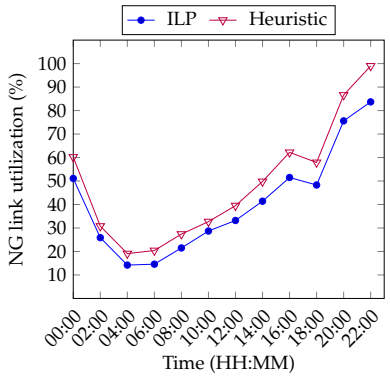


FIGURE 4.7: NG-link (base station-to-AP-to-5GC) utilization (Scenario 2).

Average end-to-end latency: Fig. 4.8 compares the average user-to-SFC end-to-end delay for ILP and heuristic for Scenario 2 experiments. Like we already discussed, UEs belonging to different clusters share some VNFs. The ILP produces an optimal solution by placing such VNFs at 5*gc.mec* nodes to minimize the overall user-to-SFC delay, but the heuristic initially places such VNFs on *gnb.mec* nodes, and therefore some UEs take the longer path (e.g., from cluster 1 to cluster 2) to access their SFC resulting in increased latency. Therefore, ILP performs better than heuristic in terms of average end-to-end delay between, as seen in Fig 4.8.

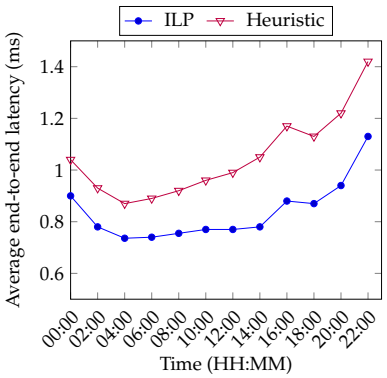


FIGURE 4.8: Average D_{E2E} based on the predicted number of UPFs from the MLP classifier model (Scenario 2).

Execution time: The above ILP formulation took 44 hours to associate 300 UEs including their latency-sensitive SFC requests composed of a number of VNFs on a mobile network comprised of six base stations, two aggregation points, and one 5G core. Therefore, we proposed a heuristic algorithm that performs a comparable association and mapping in a couple of seconds except with sub-optimal outcomes. Both ILP and heuristic were solved using ILOG CPLEX solver on an Intel Core i7 laptop with 3GHz CPU and 16 GB RAM.

4.6 Discussion

We solve a joint UE association and SFC placement problem aiming to minimize the overall user-to-**sfc!** (**sfc!**) end-to-end latency. We have seen that the ILP improves QoS of all UEs by initially placing their SFCs in ME hosts closer to base stations (*gnb.mec*) and thereby reducing *NG* backhaul link usage. Once the *gnb.mec* node CPU resources are depleted, near-real-time and non-real-time SFCs are moved/placed in ME hosts closer to aggregation points and 5G Core which results in increased usage of *Xn* and *NG* backhaul links. We evaluated the proposed model using simulations based on real-operator network topology and real-world latency values. Our results show that the average end-to-end latency reduces significantly when SFCs are placed at the ME hosts according to their latency and data rate demands. Furthermore, we propose an heuristic algorithm to address the issue of scalability in ILP, that can solve the above association/mapping problem in seconds rather than hours.

Chapter 5

AI-driven MEC Orchestration Platform

This chapter introduces two proof-of-concept implementations that enable hosting of MEC applications and performing MEC applications' autonomous lifecycle management.

(i) In Section 5.1, we present *lightMEC* prototype implementation that leverages lightweight virtualization technologies such as Dockers, Containers, and Click unikernels. We further validate our MEC platform by performing real experiments with content caching as a use case.

(ii) In Section 5.2, we present AI-driven Kubernetes-based orchestration prototype implementation for horizontal and vertical proactive auto-scaling, based on both centralized and federated DL approaches, leveraging our *lightMEC* platform. We then compare the benefits of proactive auto-scaling approach to reactive auto-scaling.

This chapter is based on two of our publications - [90] and [73].

5.1 A Vendor-agnostic MEC Platform

5.1.1 Overview

In this section, we examine the potential options where the ME host can be deployed within the mobile network. Furthermore, we discuss the main benefits and design challenges associated with each option that needs to be considered before the commercial deployment of MEC. We determine the best option for our use case based on several factors such as performance, required resources, cost and physical deployment constraints. We then present our MEC platform called *lightMEC* [90], which was developed from scratch as part of this thesis work, by focusing on the implementation details (code accessible from [91]).

5.1.2 System Design and Architecture

5.1.2.1 Deployment Options

Fig. 5.1 depicts all three possible deployment options considered by ETSI in [92].

The first option is to deploy the ME host directly at the base station. In this option, since the ME host is in close proximity to the end users, it results in very low end-to-end latency meeting the requirements of real-time applications. However, since computational power and storage capacities are limited at the base station site, only applications requiring low compute and storage resources are served by the ME host collocated directly with the base station.

The second option is to deploy the ME host at cell aggregation points. In this option, since each ME host serves a cluster of base station's, the deployment cost is significantly reduced. Also, with the possibility of having high compute and storage capacities at aggregation points, the ME host can serve high demanding applications with ease. However, there is a trade-off between low-latency and deployment cost. From the system design point of view, first and second options are considered as Bump-in-the-wire (BITW) approaches, where the ME host is placed on the Long Term Evolution (LTE) S1-interface connecting the base station and the core network.

The third option is to deploy the ME host at the edge of the core network. While this could result in a higher latency, with the recent advances in virtualization technologies, ETSI is also considering to include all or part of the core network functions as VNFs together with the ME host which is placed on the LTE user plane interface (SGi) connecting the mobile network to the external packet data network.

One of the key functionalities of the ME host is to steer IP packets from base station/core network mobile network entities to MEC applications running on the ME host. The MEC applications can either terminate the IP packets by itself (end-point mode) or modify the IP packets and pass it back to the original PDN connection (pass-through mode). In BITW approach, since S1 user plane traffic is GTP encapsulated, the ME host has to monitor S1 control plane messages to maintain the UE context information and perform GTP decapsulation/re-encapsulation and routing operations on S1 user plane packets. However, this stateful operation can be performed requiring

no modifications to either base station or core network protocol stacks which will be discussed later in detail.

Our focus being mobile edge caching in heterogeneous networks, we consider placing the ME host at cell aggregation sites/multi-RAT nodes. The decision is driven by the design modularity of BITW approach and on the possibility of caching high-definition videos considering the high storage capacity that can be made available at cell aggregation sites.

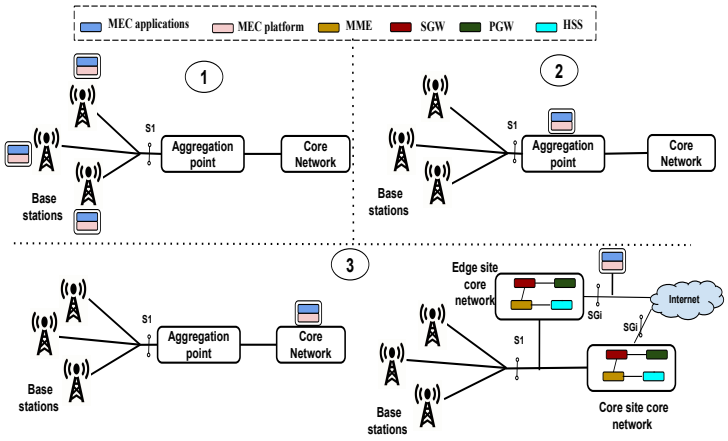


FIGURE 5.1: ME host deployment options.

5.1.2.2 System Architecture

Fig. 5.2 depicts the *lightMEC* system architecture and its functional elements that comprise the mobile edge system. On a broader context, the mobile edge system consists of the mobile edge host and the mobile edge management that facilitates to run mobile edge applications. It is to be noted that the proposed architecture is in-line with the ETSI MEC reference architecture [59].

Mobile Edge Host. This entity contains a mobile edge platform and a virtualization infrastructure built on Docker Containers and Click unikernels which provides lightweight compute, storage and networking resources for running mobile edge applications. The virtualization infrastructure also includes an OpenFlow virtual switch to route traffic among 3GPP network elements, the mobile edge services, and the mobile edge applications. The mobile edge services running on the mobile edge platform are realized using the Click Modular Router [93] and are here referred to as Light Virtual Network Functions (LVNFs), the details of which are discussed in the next subsection. The mobile edge applications are running as individual containers and can interact with mobile edge

platform to consume the services being hosted on it. The applications can range from caching to video transcoding to deep packet inspection.

Mobile Edge Management. This entity includes the virtual infrastructure manager, the mobile edge platform manager and the orchestrator. Kubernetes [30] is used as a platform for managing container-based VNFs (instantiated in the mobile edge host) while 5G-EmPOWER [94] and lightMANO [94] are used as, respectively, mobile edge platform manager and orchestrator. The mobile edge platform manager is responsible for interacting with the backhaul controller through an Intent-based networking interface and to deploy the mobile edge services LVNFs within the mobile edge host. The orchestrator has a global view of the mobile edge system regarding supported mobile edge services, available infrastructure resources, and the topology. Information about all supported LVNFs and mobile edge services are maintained in the application specific VNF catalogue. The orchestrator, depending on the application instantiation/termination request received from the Operations Support System (OSS) of an operator prepares the mobile edge platform manager and the virtualization infrastructure manager to allocate resources, to deploy services and to handle mobile edge applications.

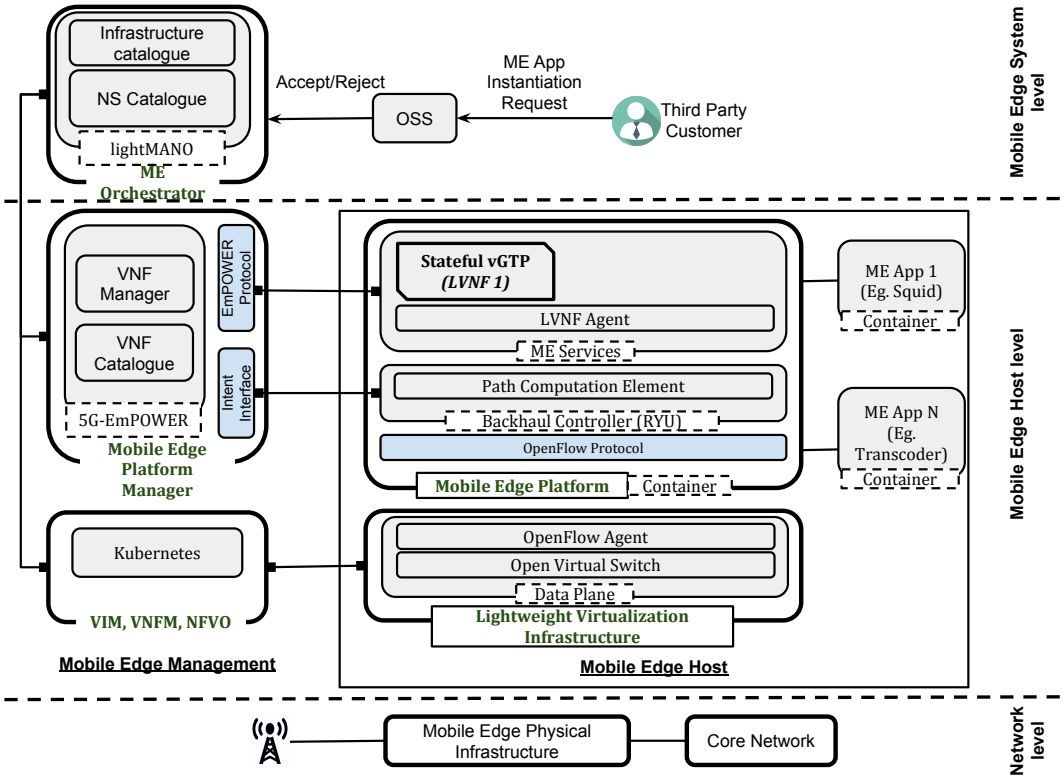


FIGURE 5.2: *lightMEC* System Architecture.

5.1.2.3 UE Context Management

We now describe two important functions of LTE i.e., session management and mobility management, to illustrate how the ME services extract the UE context information from LTE control plane messages during UE attach and handover procedures in a transparent manner. This function is fundamental in order to implement the BITW MEC deployment option used in this work. We remind the reader that this option requires to put the ME host on the S1 interface between the base station and the core network. In order to allow ME Applications to access the inner IP traffic the GTP header must be removed. However, traffic flowing from the ME Applications to the end user must be GTP encapsulated in order to be properly processed by the base station.

Session Management. Once cell search and radio synchronization procedures are performed, the UE sends an *Attach Request* message to the base station as an initial step in the UE registration procedure (see Fig. 5.3). The base station embeds this message within an *Initial UE Message* message, which consists of a unique base station-UE-S1AP-ID assigned by the base station to identify the UEs within the base station over S1-interface, and then forwards it to the MME. The ME host snoops this message and retrieves *eNodeB-UE-S1AP-ID* identifier.

Once the MME acquires International Mobile Subscriber Identity (IMSI) from the *Attach Request* message, it performs several UE authentication and security procedures with the support of HSS. If successful, the Mobility Management Entity (MME) embeds an *Attach Accept* response message within an *Initial Context Setup Request* message (see Fig. 5.4). This message includes Software Gateway (SGW) GTP tunnel information and a unique MME-UE-S1AP-ID to identify the UEs in MME over S1-interface, and then forwards it to the base station. The ME host retrieves eNodeB-UE-S1AP-ID, MME-UE-S1AP-ID, UE IP address, SGW Tunnel End Point ID (TEID) and SGW IP address information from this message to maintain UE context information for further processing. If the eNodeB-UE-S1AP-ID identifier in *Attach Request* matches to that in *Attach Accept*, the UE is added to the list of UE-associated logical S1-connections in the ME host. The base station then forwards the *Attach Accept* message towards the UE.

The UE now sends an *Attach Complete* message to the base station, which embeds this message in an *Initial Context Setup Response* message that includes base station GTP tunnel information, and forwards it to the MME (see Fig. 5.5). The ME host retrieves eNodeB-UE-S1AP-ID, MME-UE-S1AP-ID, base station TEID, and base station IP address

information from this message and updates the UE context with this new information. Completing this step establishes an S1 GTP-U tunnel for the UE to exchange uplink/downlink traffic with the mobile network.

The ME host now has the complete UE context information required to perform stateful decapsulation and re-encapsulation of GTP tunnels for routing UEs IP traffic between base station, core network and MEC applications.

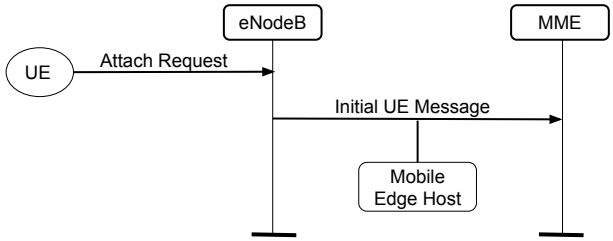


FIGURE 5.3: UE Attach Request.

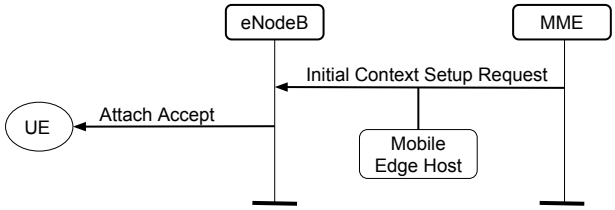


FIGURE 5.4: UE Attach Accept.

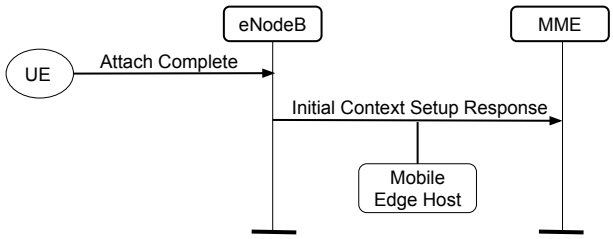


FIGURE 5.5: UE Attach Complete.

Mobility Management. After the UE is attached to the network, it periodically sends serving cell and non-serving cell measurement reports to its base station. Based on the measurements, the source base station may decide to handover the UE to another base station (X2-handover) by sending an *Handover Request* message to the target base station (see Fig. 5.6). The target base station now allocates the necessary radio resources for the

UE and responds with an *Handover Ack* message to the source base station. On the other hand, the target base station sends a *Path Switch Request* message to the MME asking it to prepare the new radio bearers. The eNodeB-UE-S1AP-ID of the UE to be handed over, the target base station IP and the target base station TEID information contained in this message are retrieved by the ME host for further processing.

The MME after receiving the *Path Switch Request* message, orders the SGW to establish new radio bearers based on the information received about the target base station GTP tunnel. Once the bearers are established, the MME sends an *Path Switch Ack* message containing eNodeB-UE-S1AP-ID and MME-UE-S1AP-ID identifiers to the target base station (see Fig. 5.7). The ME host checks if the identifiers belong to the concerned UE and if so, updates the UE context with new base station tunnel endpoint information.

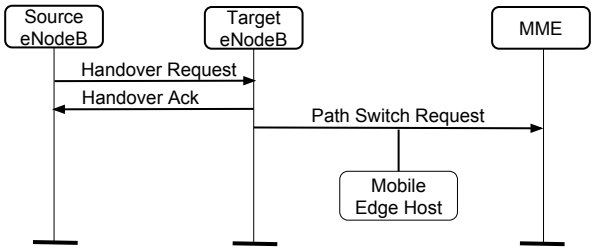


FIGURE 5.6: Path Switch Request.

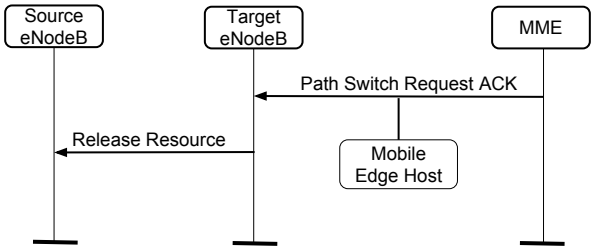


FIGURE 5.7: Path Switch Request Ack.

5.1.3 Prototype Details

We developed a prototype implementation of *lightMEC* and deployed it over an open source LTE testbed. The RAN part comprises the 3GPP-complaint LTE stack provided by the srsLTE project [95] while as core network we use nextEPC [96]. Both base station and core network nodes are running on Intel NUC boxes equipped with dual-core Intel i7 Kaby-Lake CPU, 16GB RAM and 256GB storage. Considering that the *lightMEC*

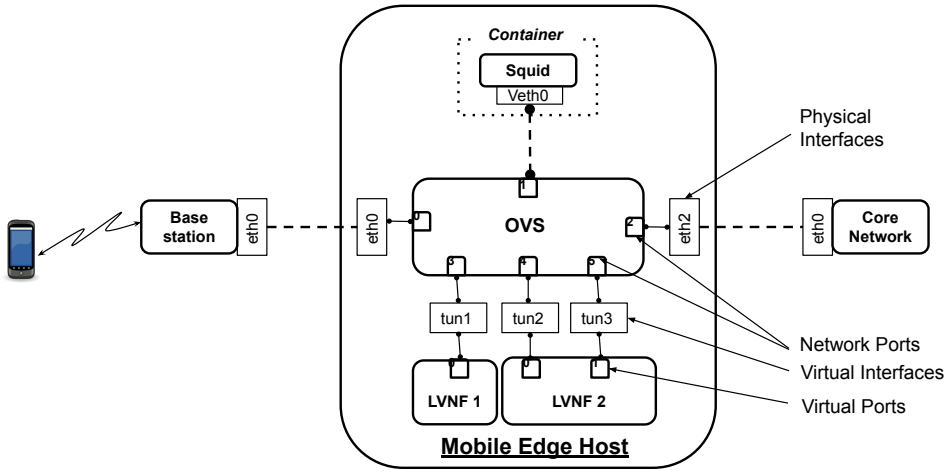


FIGURE 5.8: Mobile Edge Platform.

platform intercepts only the standardized 3GPP messages (i.e., session management and handover management) between the base station and the core network for its operation, the *lightMEC* platform is considered to be vendor-agnostic that can be used with any combination of base station/core network components.

To deploy the ME host node we use a combination of Intel NUC and Soekris 6501 boards. The LVNF agent, the Ryu SDN controller, and the caching application are all deployed as containers within a dedicated edge node. The lightMANO orchestrator, the 5G-EmPOWER controller, and the Kubernetes orchestrator are running on standard Linux machines with no particular hardware restrictions.

Squid [97] is used as ME Application. Squid is an open-sourced caching and forwarding web proxy that primarily supports HTTP, HTTPS and FTP traffic. In this experiment, Squid is configured as a transparent caching proxy i.e., all outgoing HTTP/HTTPS requests are intercepted by Squid and the corresponding responses are then cached, without requiring any changes in the client.

Fig. 5.8 depicts the internal structure of the mobile edge platform. As it can be seen it consists of a virtual software switch, the Ryu controller, the ME application, and the Stateful User Plane LVNF. This LVNF has three virtual ports with each connected to one virtual interface, which in turn is connected to one network port of the virtual software switch. The packet processing elements of click script used for this LVNF is depicted in Fig.5.9. Port 0 of LVNF receives LTE control plane traffic (Stream Control Transmission Protocol (SCTP)) which is then passed to the S1APMonitor element that extracts the

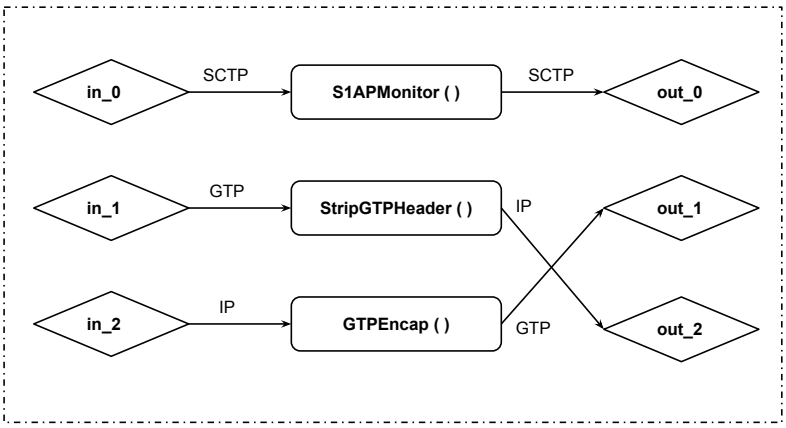


FIGURE 5.9: Stateful User Plane LVNF.

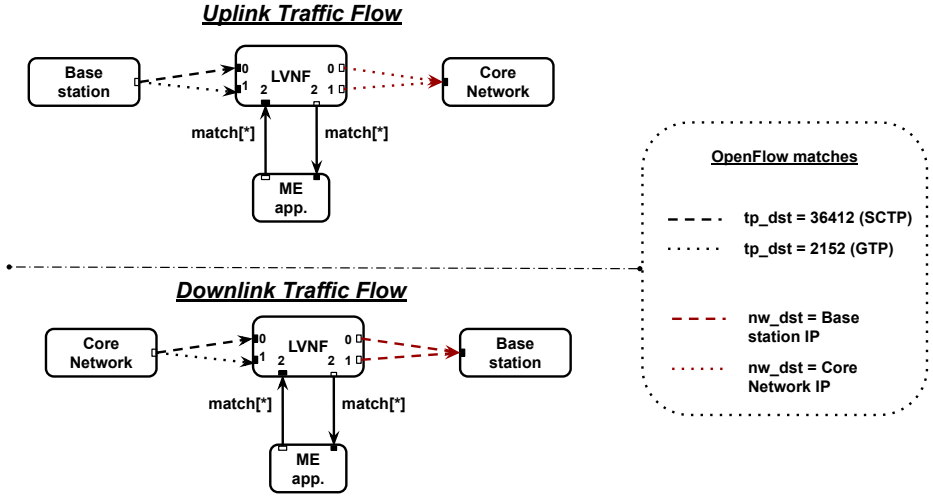


FIGURE 5.10: Stateful User Plane LVNFs chain illustrating traffic flow.

UE context information. Port 1 of LVNF receives LTE user plane traffic (GTP) which is then passed to the StripGTPHeader element that removes the GTP header and forwards IP packets on output port 2. Port 2 of LVNF receives IP traffic from applications such as squid (caching), proxy servers, etc., which is then passed to GTPEncap element that performs GTP re-encapsulation and forwards GTP packets on output port 1.

Fig.5.10 illustrates the virtual connection points between the LVNF and other network elements implemented by means of OpenFlow rules in Open Virtual Switch (OVS) switch. The control plane traffic (SCTP) from base station/core network is directed to port 0 of LVNF, the user plane traffic (GTP) from base station/core network is directed to port 1 of LVNF and any

other IP traffic from the ME application is directed to port 2 of LVNF. The traffic matching is performed by the openflow rules configured in OVS switch by Ryu controller. The output traffic from LVNF is forwarded to appropriate destination points depending on the destination IP address of the packets.

5.1.4 Performance Evaluation

To illustrate the potential of our approach, we measure three network performance metrics, namely: MEC specific latency metrics, ME service VNF metrics, and cache latency metrics.

5.1.4.1 MEC Specific Latency metrics

To assess the difference in latency performances i.e., Round-Trip-Time (RTT) (the time taken for receiving the response after the initial request was sent by the UE) between the MEC and the non-MEC deployment options, we perform simple ping tests using Internet Control Message Protocol (ICMP) messages. Fig.5.11 plots the average RTT when the ping server was located at the mobile edge node, while Fig.5.12 and Fig.5.13 represents the average RTT when the server was located in France and in the USA respectively. The experiments were performed for different packet sizes and for different Inter Departure Times (IDT). As expected, the RTT is significantly lower when the ping is answered from edge node.

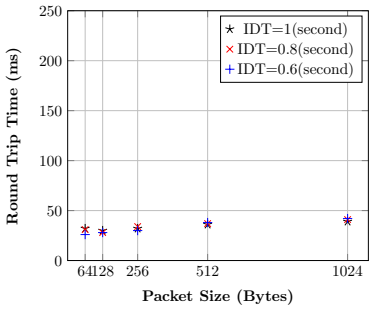


FIGURE 5.11: RTT to a server located in Edge node.

5.1.4.2 ME Service VNF metrics

We performed a set of 10 UE initiated attach events to measure the time taken by S1APMonitor element of the Stateful User Plane LVNF to detect the attach events and to extract the UE context information. The average

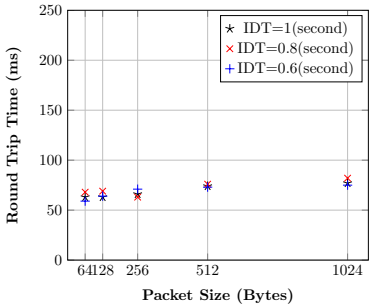


FIGURE 5.12: RTT to a server located in France.

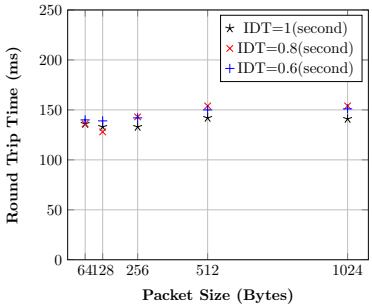


FIGURE 5.13: RTT to a server located in USA.

processing time was 1.4*ms*. We then generated IP traffic from the UE to measure the time taken by GTPEncap/StripGTPHeader element of the LVNF to perform encap/decap operations. The average processing time was 30μs when analysed over a sample of 100 packets.

5.1.4.3 Cache Latency metrics

We performed an experiment to measure the difference in RTT when the user is served by the caching application running on the mobile edge node, instead of the original web server. The measurements were carried out by making web requests to five popular web pages. The difference in latency is clearly evident from Fig. 5.14.

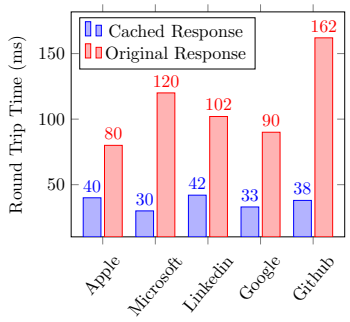


FIGURE 5.14: Caching latency metrics.

5.1.5 Discussion

5G is expected to facilitate innovative low-latency applications to enhance the user experience, and it will (mostly) do so by using existing 4G network infrastructure. With video service dominating the mobile network traffic it is becoming challenging for network operators to avoid network congestion in mobile networks and to guarantee Quality of Experience (QoE) for mobile users. This work tried to address these problems by proposing a novel MEC architecture that leverages SDN and NFV technologies to reduce the barrier for deploying MEC applications and services with existing LTE installations. We proposed a modular MEC architecture, *lightMEC*, by describing the functionality of each element in the node. A proof-of-concept implementation of *lightMEC* was also introduced and validated in a real test-bed with content caching as a use case. As a future work, we plan to extend *lightMEC* platform to support additional MEC services further enabling more innovative applications.

5.2 AI-driven Kubernetes Orchestration Platform

5.2.1 Overview

The Mobile Edge hosts have a limited amount of physical and virtual resource capacity compared to the cloud data centers. Therefore, it is necessary to manage these resources efficiently. An essential characteristic of the NFV is elasticity. While NFV Management and Orchestration (MANO) entity (e.g.,Kubernetes) enables VNFs to dynamically obtain and release resources according to the varying demands, choosing the correct amount of resources is not a simple task. Current virtualization platforms offer autoscaling capabilities using a manual trigger that is reactive (e.g., if CPU utilization reaches 80%, scale-up VNF by one). However, it would be beneficial to have a predictive autoscaling mechanism in NFV MANO that could beforehand automatically adapt the resources to the workload managed by the VNF without any human intervention. In the rest of the section, we introduce the AI-driven Kubernetes orchestration prototype that we implemented, as part of this thesis, by leveraging our *lightMEC* platform and assess the performance of the proposed deep learning models (from sections 3.2 and 3.3) in a practical setup.

5.2.2 System Design and Architecture

5.2.2.1 Kubernetes-based Orchestration

Docker [98] is a software platform designed to build, deploy, and manage applications easily, quickly, and efficiently. A Docker container image is a standalone, lightweight, executable software package, which includes the application code and all the necessary data for its execution (e.g., dependencies, system libraries and tools, configuration files). Therefore, containerized applications can be run reliably in different computing environments. Docker consists of a container-runtime (e.g., docker-engine) that enables us to create and run container images, either utilizing Representational state transfer (REST) APIs or command-line interface. Docker also allows for configuring a container with a resource quota (e.g., CPU) that it can use on the host machine. The resource quota can be updated on runtime, thus also enabling vertical scaling.

Kubernetes [30] is a container-orchestration system for simplifying and automating application deployment, scaling, migration, and management across a distributed cluster of Docker-enabled nodes (e.g., ME Host nodes, cloud nodes). A pod is the basic and smallest execution unit of an application within the Kubernetes object model that we can create/deploy. A pod comprises one or multiple application containers, storage resources, and policies on how the container must run. Kubernetes architecture follows the master-workers pattern where the master deals with the orchestration and scheduling of pods in the worker nodes based on their computational capabilities. The default scheduling policy is *Spread*, which distributes pods among all worker nodes. To scale an application horizontally, we must run multiple pods, i.e., one for each instance, which, in Kubernetes, is referred to as replication.

5.2.2.2 AI-driven Kubernetes-based Orchestration

To enable proactive autonomic capabilities in Kubernetes orchestration, we extend the Kubernetes architecture to introduce the MAPE closed control loop. The *Monitor* component collects operational data (e.g., CPU), either through a centralized or distributed approach, from all deployed pods in the worker nodes. The *Analyze* component uses the collected data to perform intelligent analytics (e.g., DL for proactive vMAF auto-scaling), either through centralized or distributed techniques, and to decide whether an adaptation is necessary for the deployed pods. If the adaptation is

required, the *Plan* component defines an adaptation plan (e.g., scale vMAF1 by one instance at time $t + i$) for the deployed vMAFs, which is performed by the *Execute* component, i.e., Kubernetes master. The modularity and the support of APIs allow us to integrate our MAPE components in Kubernetes easily. To exchange data and model updates between central server (Master) and local ME hosts (Workers), we use websocket server-client connections. In practical deployments, a distributed streaming platform (e.g., Apache Kafka **kafka**) could be used for building real-time data pipelines instead of websocket connections.

5.2.3 Prototype Details

Our AI-driven Kubernetes orchestration prototype follows the masters-workers pattern, which decentralizes the MAPE closed control loop. In particular, as shown in Fig. 5.15, the prototype includes one master node, which runs all the *MAPE* phases, and three ME host worker nodes, which runs either the *M* phase for centralized analytics approach or *MAP* phases for distributed analytics approach. The nodes are interconnected using a Flannel overlay network, which facilitates cluster networking. Each of these nodes runs a Kubelet, i.e., a Kubernetes node agent that plays the role of a *Monitor* component, which natively collects metrics (e.g., CPU utilization) about containers running on the node in the time-series format. The collected data is either transferred to a centralized master node or stored in the respective worker nodes. Centralized and Federated DL takes the role of *Analyze* and *Plan* components, which analyzes/interprets the collected time-series data, either through centralized or federated approaches, and predicts the required future actions. Finally, Kubernetes Master (i.e., kube-scheduler) serves the role of *Execute* component, which increases or decreases the vMAF pods (i.e., through replica sets) or 100 millicore CPU units per vMAF (i.e., by updating the CPU capacity of the pod with a new deployment) in a particular worker node. For our demonstration, nginx web servers are containerized and deployed as MEC application pods on worker nodes.

For reactive auto-scaling, the Metrics Server [99] retrieves metrics exposed by kubelet on each node through the Resource Metrics API and aggregates the cluster-wide resource usage data. The Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) are custom resource definition objects (i.e., resource quotas such as CPU, memory) for horizontal

and vertical auto-scaling, respectively. These autoscaler objects fetch metrics from a series of aggregated APIs provided by Metrics Server and thus facilitates auto-scaling the number of pods or CPU millicores.

Moreover, the Prometheus monitoring application [100] can retrieve all the collected metrics from the worker nodes into a time-series database. It then allows querying them on-demand, thus facilitating data visualization through Grafana [101] integration.

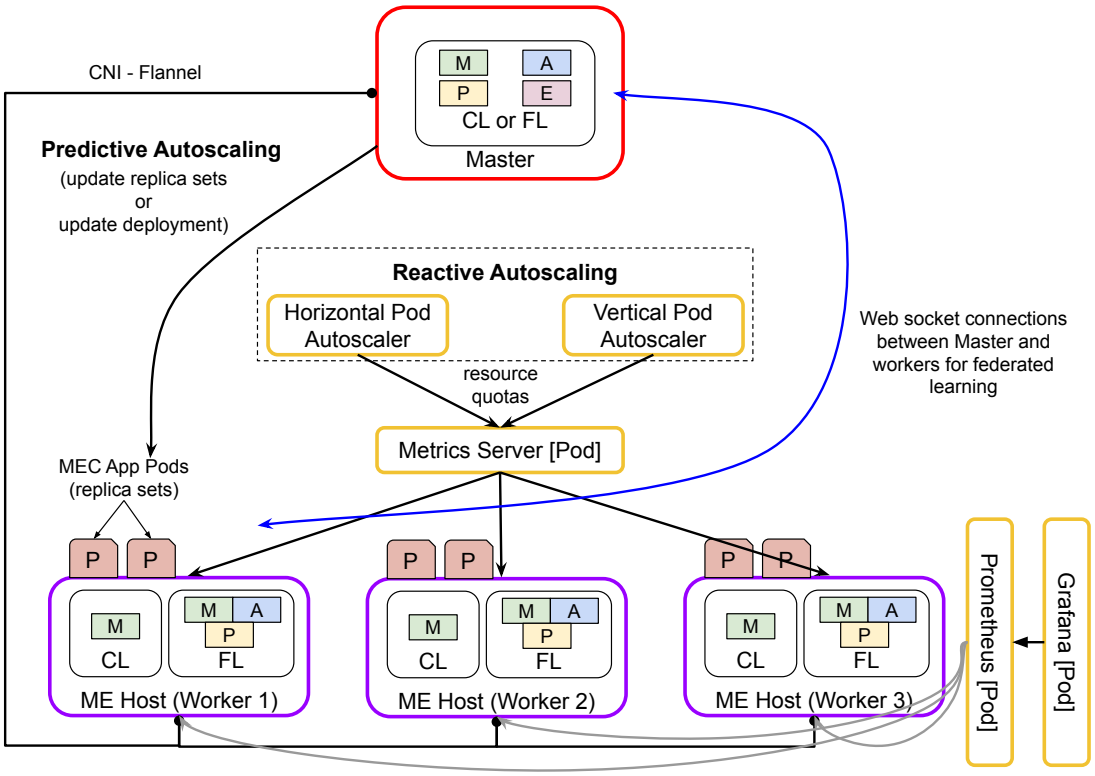


FIGURE 5.15: Prototype of AI-driven Kubernetes Orchestration in MEC-enabled Mobile Networks.

5.2.4 Performance Evaluation

Based on the encouraging simulation results (sections 3.2 and 3.3), we evaluate both the centralized and federated learning algorithms in the AI-driven Kubernetes orchestration prototype that we described earlier. Furthermore, we compare the native reactive auto-scaling solution against AI-driven proactive auto-scaling solutions. The reference MEC application used is the Nginx web server, which, upon request, serves the web page request. As seen in Fig. 5.16, the MEC application receives a varying number of concurrent requests, such that the incoming workload pattern

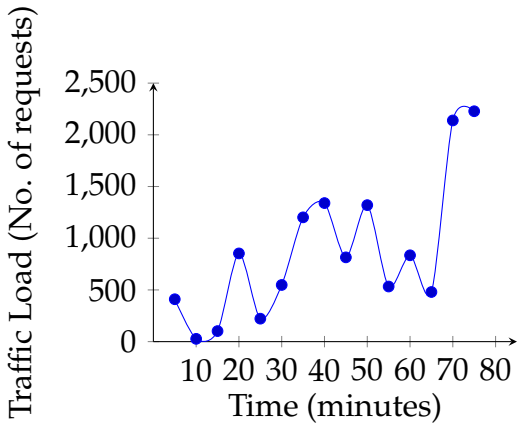


FIGURE 5.16: Workload used in the prototype experiments.

follows our traffic load dataset used in the simulation. However, when replaying the reference dataset (from Chapter 3), the time granularity of the samples is reduced from hourly to five-minutely, for simplicity in evaluations. We measure the average target response time for different approaches of horizontal and vertical auto-scaling.

Here, we describe the Kubernetes configuration setting used for reactive auto-scaling. For horizontal auto-scaling, the HPA checks metrics values through metrics-server at 30 second time-intervals, and the relative metrics tolerance is set at 5%. The threshold levels for HPA are set at 100 millicore CPU utilization (i.e., 10% of maximum capacity) to draw parallel comparisons with vertical scaling. Additionally, the HPA waits for 3 minutes following the previous scale-up event to allow metrics to stabilize. It then waits for 5 minutes from the last scale-down event to avoid autoscaler thrashing. For vertical auto-scaling, the VPA checks metrics values through metrics-server at 10 second time-intervals and operates using *Auto* mode (i.e., not a typical reactive approach with threshold levels but uses a recommender system that can predict future pod resource requirements). The VPA waits for approximately 5 minutes following the previous scale-up or scale-down event to avoid the ping-pong effect. The changes in the resource limits of pods result in restarting the pod that might lead to instability. It is to be noted that VPA and HPA cannot work together on the same pod since both are not compatible.

5.2.4.1 Proactive vs Reactive Horizontal Auto-scaling

In the QoS-prioritized case, optimizing the average response time is more important than minimizing the resource allocation cost. Fig. 5.17 compares

the reactive horizontal auto-scaling approach with the best performing (based on our simulation results) QoS-prioritized proactive centralized learning neural network model (LSTM). Minimizing resource allocation cost is more important than optimizing the average response time in a cost-prioritized case. Fig. 5.18 compares the reactive horizontal auto-scaling approach with the best performing (based on our simulation results) cost-prioritized proactive centralized learning neural network model (CNN-LSTM). The bottom plot represents the number of containers with respect to time, and the upper plot represents the corresponding average round-trip-time at that particular time instance.

Based on both the figures (Fig. 5.17 and Fig. 5.18), it is evident that reactive auto-scaling (i.e., orange lines) is slow in reacting to sudden traffic load spikes compared to proactive auto-scaling (blue lines). Therefore, the number of scaled containers at a particular time instance is lesser/more than the actual requirement (i.e., black lines), which directly increases the overall RTT for the MEC application users. On the other hand, in cost-prioritized auto-scaling, the number of scaled containers is less than that of the QoS-prioritized auto-scaling to handle the same traffic load, which is also the reason for the increased round trip time that we observe when comparing both the figures. Hence, the trade-off between cost and QoS is evident in both approaches.

5.2.4.2 Centralized vs Federated Proactive Auto-scaling

Fig. 5.19 and Fig. 5.20 compares the performance of centralized learning model to federated learning models (i.e., with and without model averaging), trained on the distributed data (i.e., 3 physical ME host nodes) using CNN-LSTM neural network, for QoS-prioritized horizontal and vertical auto-scaling, respectively. The bottom plot represents the number of containers or the number of 100 millicore CPU units with respect to time. The upper plot represents the corresponding average round-trip-time at that time instance.

Based on both the figures (Fig. 5.19 and Fig. 5.20), it is clear that the CL model (blue lines) performs better auto-scaling predictions compared to FL models, i.e., with (red lines) and without (green lines) model averaging. Consequently, the overall round trip time for the MEC application users increases in the FL models approach. In particular, for vertical auto-scaling (Fig. 5.20), RTTs are significantly higher compared to the horizontal auto-scaling, especially at traffic load peaks, due to the pod restarts on each

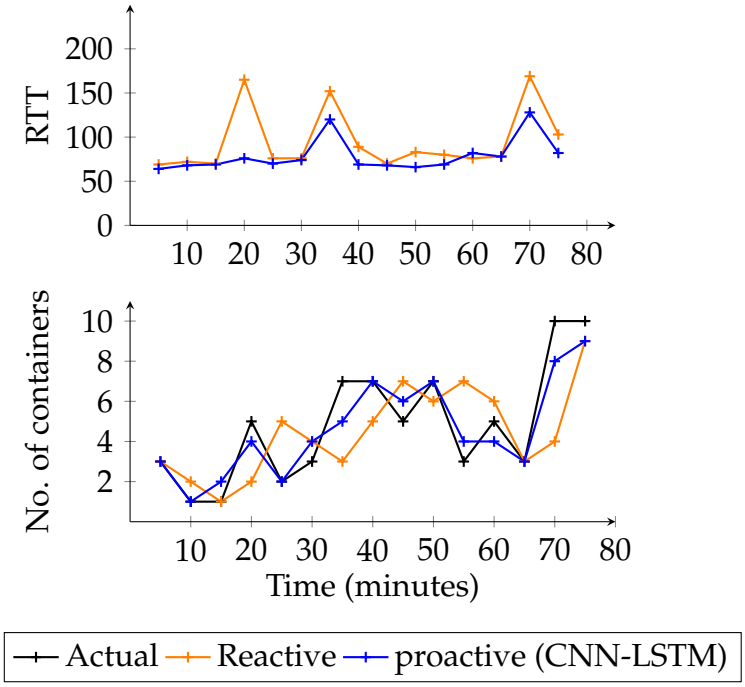


FIGURE 5.17: Comparision of reactive and best performing proactive QoS-prioiritized centralized learning model (CNN-LSTM) for horizontal auto-scaling.

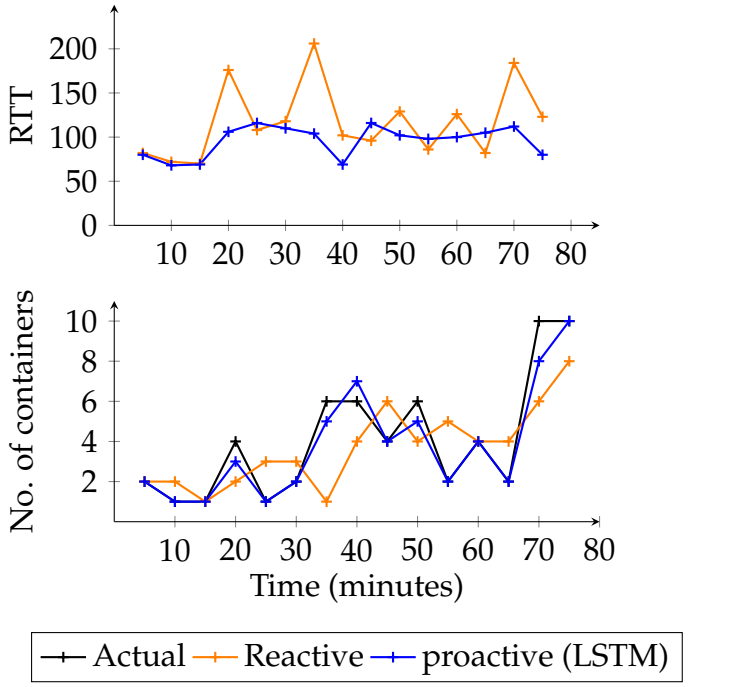


FIGURE 5.18: Comparision of reactive and best performing proactive Cost-prioiritized centralized learning model (LSTM) for horizontal auto-scaling.

adaptation and the time it takes to bring the Kubernetes cluster to a stable state.

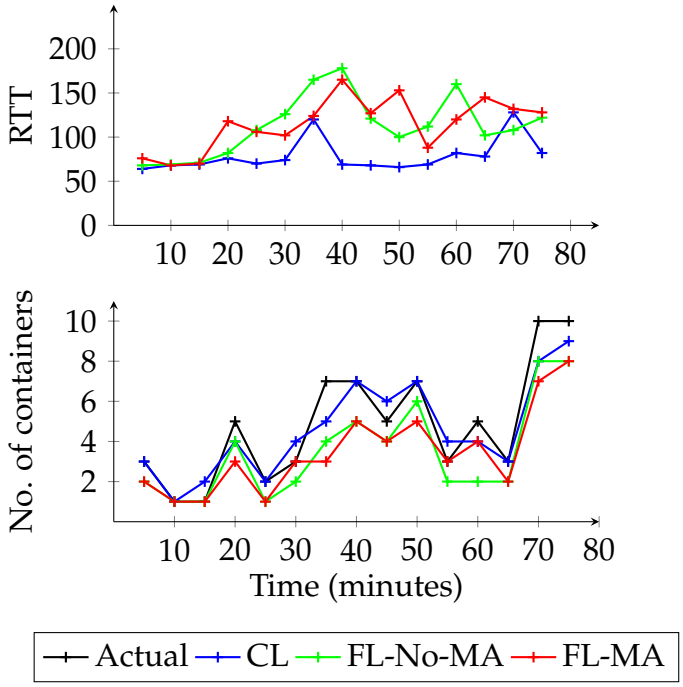


FIGURE 5.19: Comparision of centralized, FL-No-MA and FL-MA proactive QoS-prioiritized CNN-LSTM models for horizontal auto-scaling.

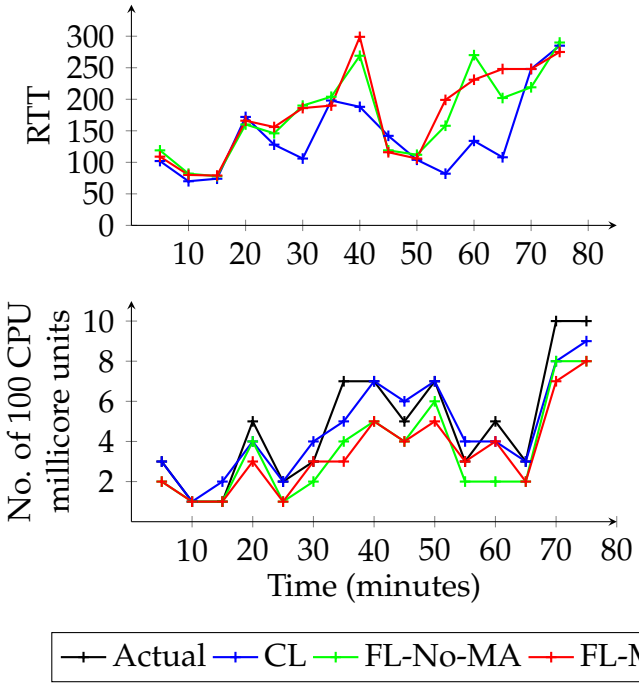


FIGURE 5.20: Comparison of centralized, FL-No-MA and FL-MA proactive QoS-prioiritized CNN-LSTM models for vertical auto-scaling.

5.2.5 Discussion

We designed and implemented a AI-driven Kubernetes-based orchestration prototype by leveraging our MEC platform. We evaluate both centralized and federated learning models for horizontal and vertical containerized-vMAF (i.e., Nginx webserver as a MEC application) auto-scaling by measuring the overall round trip time. Furthermore, we compare the native reactive auto-scaling approach to our proactive auto-scaling techniques. The prototype evaluations confirm the simulation results achieved in the first part.

Chapter 6

Conclusions and Future Works

Every new generation of mobile technology tries to build on the previous one. 1G allowed users to make calls wirelessly over analogue voice networks. 2G broadened the offered services by introducing digitization, mobile roaming and short text messaging. 3G facilitated data and connection to the internet, which was valuable in the smartphone age. With 4G, users saw notable improvements in the internet connection (i.e., speeds up to 1 gigabit per second) that allowed for uploading larger data files. 5G builds upon 4G to achieve the performance targets set by ITU set in 2015, which includes eMBB, URLLC and mMTC on a shared common mobile network infrastructure. The telecom industry has identified NFV, MEC, SDN and AI as some of the key technology enablers although to achieve the vision of 5G and beyond although several challenges need to be solved before reaching its full potential. In this dissertation, we identify three challenges to realize the vision of 5G and beyond and provide solutions to each of them.

Chapter 1 presents the motivation and objectives, discusses the main problem statements, and provides the dissertation's outline to set the stage for the rest of the chapters.

Chapter 2 provides a background on 5G mobile networks, the various cloud computing technologies, NFV MANO and AI. Furthermore, we discuss the related works in VNF auto-scaling, VNF placement, SFC placement, user association and existing MEC orchestration platforms and highlight why our work is different from the existing literature.

Chapter 3 aims at applying ML techniques to VNF auto-scaling. Firstly, we proposed two neural-network based MLP models (i.e., a classifier and a regressor) to facilitate proactive auto-scaling of VNFs, based on the traffic traces obtained from a commercial operator. We evaluated the proposed models for its effectiveness in accurately predicting the amount of VNF instances required as a function of the network traffic it should process. For

MLP classifier, we measured accuracy, precision, recall, F-measure, and finally reported confusion matrix, while for MLP regressor we measured MSE, MAE, RMSE, and R^2-score. Our results show that both MLP classifier and MLP regressor models have strong predicting capability for auto-scaling. However, MLP regressor outperforms MLP classifier in terms of accuracy. Secondly, we design centralized DL techniques for predictive VNF auto-scaling with QoS-prioritized and cost-prioritized objectives. We model the auto-scaling problem as a time series forecasting problem and determine one-step and multi-step future predictions using real-operator traffic load datasets for training, validation, and testing. We evaluate and compare ANN, LSTM, and CNN-LSTM models by measuring the MAE, MSE, and RMSE key performance metrics. For centralized learning and one-step predictions, CNN-LSTM performs the best for the QoS-prioritized objective and LSTM performs the best for the cost-prioritized goal. For centralized learning and multi-step predictions, the encoder-decoder CNN-LSTM model outperforms the encoder-decoder LSTM model. Thirdly, we design federated DL techniques for predictive VNF auto-scaling with QoS-prioritized and cost-prioritized objectives as a time series forecasting problem. For federated learning, both LSTM and CNN-LSTM models perform equally better than the ANN model. FL performs poorly compared to centralized learning due to the non i.i.d. data samples in the individual local ME host nodes. The detailed analysis of the achieved results are discussed in section 3.1.5, section 3.2.5 and section 3.3.5.

Realizing VNF scaling has been challenging, and solutions to date have not been entirely successful. The main hurdle is that many VNFs are stateful, with the state that may be read or updated quite often (e.g., per-packet, per-flow). Consequently, VNF scaling requires more than just spinning up a new container and updating the load-balancer to send a portion of traffic to it. Instead, VNF scaling must also migrate states across instances and guarantee affinity between packets and their state (i.e., a packet being directed to the VNF instance that holds the state needed to process that packet). Therefore, ML-based proactive stateful VNF autoscaling and ML-based proactive stateful VNF migration are two of the research topics that are still open to be further investigated.

In Chapter 4, we solve a joint UE association and SFC placement problem aiming to minimize the overall user-to-sfc end-to-end latency. We have seen that the ILP improves QoS of all UEs by initially placing their SFCs in MEC nodes closer to gNodeBs and thereby reducing *NG* backhaul

link usage. Once the gNodeB MEC node CPU resources are depleted, near-real-time and non-real-time SFCs are moved/placed in MEC nodes closer to aggregation points and 5GC which results in increased usage of Xn and NG backhaul links. We evaluated the proposed model using simulations based on real-operator network topology and real-world latency values. Our results show that the average end-to-end latency reduces significantly when SFCs are placed at the MEC nodes according to their latency and data rate demands. Furthermore, we propose an heuristic algorithm to address the issue of scalability in ILP, that can solve the above association/mapping problem in seconds rather than hours. The detailed analysis of the achieved results are discussed in section 4.5.2.

Reliability is one of the major challenge in NFV-based 5G networks which can lead to significant revenue loss and customer dissatisfaction. Normally, redundancy is used to increase the reliability of network services. However, redundancy requires additional resources and thus increases the capital and operational cost significantly. Therefore, reliability-aware SFC placement is an open research topic that needs to be investigated to provision reliable communication services with minimal overhead on the network resource consumption.

Chapter 5 proposes a novel MEC architecture that leverages SDN and NFV technologies to reduce the barrier for deploying MEC applications and services with existing LTE installations. We proposed a modular MEC architecture, lightMEC, by describing the functionality of each element in the node. A proof-of-concept implementation of lightMEC was also introduced and validated in a real test-bed with content caching as a use case. The second part of the chapter aims to design and implement a AI-driven Kubernetes-based orchestration prototype by leveraging our lightMEC platform. We evaluate both centralized and federated learning models for horizontal and vertical vMAF (i.e., Nginx web server as a MEC application) auto-scaling by measuring the overall round trip time. Furthermore, we compare the native reactive auto-scaling approach to our predictive auto-scaling techniques. The prototype evaluations confirm the simulation results achieved in Chapter 3. The detailed analysis of the achieved results are discussed in section 5.1.4 and section 5.2.4.

To maintain large-scale containerized MEC applications serving thousands of users, a single Kubernetes cluster is not sufficient. Therefore, multi-cluster Kubernetes deployments are necessary to provide an excellent user experience. With multi-clusters, users can distribute the application

across different regions, exponentially increasing the availability. The same can be applied for scaling instead of a single cluster trying to handle all the scaling requirements. Multi-cluster deployments are distributed across multiple clusters and scales according to a load of a specific cluster. Hence, end users will have a much better experience w.r.t. latency and performance while interacting with the applications. Therefore, extending our prototype to support multi-cluster kubernetes deployment is part of our future work.

Bibliography

[1] G. Forecast, "Cisco visual networking index: Global mobile data traffic forecast update, 2017–2022", *Update*, vol. 2017, p. 2022, 2019.

[2] S. Dang, O. Amin, B. Shihada, and M.-S. Alouini, "What should 6g be?", *Nature Electronics*, vol. 3, no. 1, pp. 20–29, 2020.

[3] W. Saad, M. Bennis, and M. Chen, "A vision of 6g wireless systems: Applications, trends, technologies, and open research problems", *IEEE network*, vol. 34, no. 3, pp. 134–142, 2019.

[4] "NGMN 5G Whitepaper 2", NGMN, Whitepaper, 2020.

[5] "Minimum requirements related to technical performance for IMT-2020 radio interface(s)", International Telecommunication Union, Report ITU-R M.2410-0, 2017.

[6] B. Blanco, J. O. Fajardo, I. Giannoulakis, E. Kafetzakis, S. Peng, J. Pérez-Romero, I. Trajkovska, P. S. Khodashenas, L. Goratti, M. Paolino, *et al.*, "Technology pillars in the architecture of future 5g mobile networks: Nfv, mec and sdn", *Computer Standards & Interfaces*, vol. 54, pp. 216–228, 2017.

[7] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking", *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[8] *International Telecommunication Union Telecommunication*. [Online]. Available: https://www.itu.int/en/Pages/default.aspx.

[9] *Internet Engineering Task Force*. [Online]. Available: https://www.ietf.org/.

[10] *Internet Research Task Force*. [Online]. Available: https://irtf.org/.

[11] *Broadband Forum*. [Online]. Available: https://www.broadband-forum.org/.

[12] *Metro Ethernet Forum*. [Online]. Available: https://www.mef.net/.

[13] Y. Li and M. Chen, "Software-defined network function virtualization: A survey", *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[14] *European Telecommunications Standards Institute*. [Online]. Available: https://www.etsi.org/.

[15] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration", *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[16] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges", *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[17] *3rd Generation Partnership Project*. [Online]. Available: https://www.3gpp.org/.

[18] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges", *Computer Networks*, vol. 167, p. 106 984, 2020.

[19] *ETSI Network Function Virtualization*. [Online]. Available: https://www.etsi.org/committee/nfv.

[20] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects", *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[21] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, 2. MIT press Cambridge, 2016, vol. 1.

[22] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration", ETSI GS NFV-MAN 001, 2014.

[23] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "Mlbase: A distributed machine-learning system.", in *Cidr*, vol. 1, 2013, pp. 2–1.

[24] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences", *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[25] A. Tealab, "Time series forecasting using artificial neural networks methodologies: A systematic review", *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 334–340, 2018.

[26] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications", *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[27] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey", *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[28] *IBM ILOG CPLEX Optimizer*. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer.

[29] *Linux Containers*. [Online]. Available: https://linuxcontainers.org/.

[30] *Kubernetes*. [Online]. Available: https://kubernetes.io/.

[31] B. Hayes, *Cloud computing*, 2008.

[32] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey", *Future generation computer systems*, vol. 29, no. 1, pp. 84–106, 2013.

[33] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues", in *Proceedings of the 2015 workshop on mobile big data*, 2015, pp. 37–42.

[34] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks", *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 725–737, 2015.

[35] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms", in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 161–168.

[36] Z. Ghahramani, "Unsupervised learning", in *Summer School on Machine Learning*, Springer, 2003, pp. 72–112.

[37] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey", *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[38] H. Yu, J. Yang, and C. Fung, "Fine-grained cloud resource provisioning for virtual network function", *IEEE Transactions on Network and Service Management*, 2020.

[39] S. Dutta, T. Taleb, and A. Ksentini, "Qoe-aware elasticity support in cloud-native 5g systems", in *IEEE International Conference on Communications (ICC)*, IEEE, 2016, pp. 1–6.

[40] G. A. Carella, M. Pauls, L. Grebe, and T. Magedanz, "An extensible autoscaling engine (ae) for software-based network functions", in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, IEEE, 2016, pp. 219–225.

[41] C. H. T. Arteaga, F. Rissoi, 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.

[42] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments", *Journal of grid computing*, vol. 12, no. 4, pp. 559–592, 2014.

[43] J. G. De Gooijer and R. J. Hyndman, "25 years of time series forecasting", *International journal of forecasting*, vol. 22, no. 3, pp. 443–473, 2006.

[44] J. G. De Gooijer and K. Kumar, "Some recent developments in non-linear time series modelling, testing, and forecasting", *International Journal of Forecasting*, vol. 8, no. 2, pp. 135–156, 1992.

[45] G. Bontempi, S. B. Taieb, and Y.-A. Le Borgne, "Machine learning strategies for time series forecasting", in *European business intelligence summer school*, Springer, 2012, pp. 62–77.

[46] N. K. Ahmed, A. F. Atiya, N. E. Gayar, and H. El-Shishiny, "An empirical comparison of machine learning models for time series forecasting", *Econometric Reviews*, vol. 29, no. 5-6, pp. 594–621, 2010.

[47] T. L. Duc, R. G. Leiva, P. Casari, and P.-O. Östberg, "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey", *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–39, 2019.

[48] Z. Zaman, S. Rahman, and M. Naznin, "Novel approaches for vnf requirement prediction using dnn and lstm", in *2019 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2019, pp. 1–6.

[49] I. Alawe, A. Ksentini, Y. Hadjadj-Aoul, and P. Bertin, "Improving traffic forecasting for 5g core network scalability: A machine learning approach", *IEEE Network*, vol. 32, no. 6, pp. 42–49, 2018.

[50] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions.", *Proceedigs of the 10th International Conference on Network and Service Management (CNSM)*, 2014.

[51] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, O. C. Muniz and B. Duarte, "Orchestrating virtualized network functions.", *IEEE Transactions on Network and Service Management*, 2016.

[52] S. Oechsner and A. Ripke, "Flexible support of VNF placement functions in OpenStack", in *Proc. of 1st IEEE Conference on Network Softwarization (NetSoft)*, London, United Kingdom, 2015.

[53] B. Martini, F. Paganelli, P. Cappanera, S. Turchi and P. Castoldi, "Latency-aware composition of virtual functions in 5g", in *Proc. of 1st IEEE Conference on Network Softwarization (NetSoft)*, London, United Kingdom, 2015.

[54] *Openstack*. [Online]. Available: https://www.openstack.org/.

[55] A. Alleg, T. Ahmed, M. Mosbah, R. Riggio, and R. Boutaba, "Delay-aware vnf placement and chaining based on a flexible resource allocation approach", in *13th International Conference on Network and Service Management (CNSM)*, IEEE, 2017, pp. 1–7.

[56] S. Agarwal, F. Malandrino, C.-F. Chiasserini, and S. De, "Joint vnf placement and cpu allocation in 5g", in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, 2018, pp. 1943–1951.

[57] H. Hawilo, M. Jammal, and A. Shami, "Orchestrating network function virtualization platform: Migration or re-instantiation?", in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, IEEE, 2017, pp. 1–6.

[58] ETSI, "Mobile Edge Computing (MEC);Technical Requirements", ETSI GS MEC 002, 2016.

[59] ETSI, "Mobile Edge Computing (MEC);Framework and Reference Architecture", ETSI GS MEC 003, 2016.

[60] ETSI, "Mobile Edge Computing (MEC); Service Scenarios", ETSI GS MEC 004, 2016.

[61] ETSI, "Mobile Edge Computing (MEC); Proof of Concept Framework", ETSI GS MEC 005, 2016.

[62] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing a key technology towards 5G", *ETSI White Paper vol. 11*, 2015.

[63] A. Ahmed and E.Ahmed, "A survey on mobile edge computing", in *Proc. of 10th International Conference on Intelligent Systems and Control (ISCO)*, Coimbatore, Tamilnadu, India, 2016.

[64] H. Li, G. Shou, Y. Hu, and Z. Guo, "Mobile edge computing: progress and challenges", *Proc. of IEEE MobileCloud*, 2016.

[65] J. Cho, B. Nguyen, A.Banerjee, "SMORE: Software-Defined Networking Mobile Offloading Architecture", in *Proc. of 4th workshop on All things cellular: operations, applications, and challenges*, Chicago, Illinois, USA, 2014.

[66] A. Banarjee, X. Chen, J. Erman, V. Gopalakrishnan, S. Lee, and J. Van Der Merwe, "MOCA: a lightweight mobile cloud offloading architecture", in *Proc. of 8th ACM international workshop on Mobility in the evolving internet architecture*, Miami, Florida, USA, 2013.

[67] S. Wang, et. al., "Mobile Micro-Cloud: Application Classification,Mapping, and Deployment", *Annual Fall Meeting of ITA (AMITA)*, 2013.

[68] K. Wang, M. Shen, and J. Cho, "MobiScud: A Fast Moving Personal Cloud in the Mobile Network", *Workshop on All Things Cellular:Operations, Applications and Challenge*, 2015.

[69] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-MeCloud: WhenCloud Services Follow Mobile Users", *IEEE Transactions on CloudComputing*, PP(99), 1-1, 2016.

[70] J. Liu, T. Zhao, S. Zhou, Y. Cheng, and Z. Niu, "CONCERT: a cloud-based architecture for next-generation cellular systems", *IEEE Wireless Communications*, 2014.

[71] T. Subramanya and R. Riggio, "Machine learning-driven scaling and placement of virtual network functions at the network edges", in *2019 IEEE Conference on Network Softwarization (NetSoft)*, IEEE, 2019, pp. 414–422.

[72] T. Subramanya, D. Harutyunyan, and R. Riggio, "Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks", *Computer Networks*, vol. 166, p. 106 980, 2020.

[73] T. Subramanya and R. Riggio, "Centralized and federated learning for predictive vnf autoscaling in multi-domain 5g networks and beyond", *2021 IEEE Transactions on Network and Service Management*, 2021.

[74] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities", *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, 2018.

[75] *Theano*. [Online]. Available: http://deeplearning.net/software/theano/.

[76] *Tensorflow*. [Online]. Available: https://www.tensorflow.org/.

[77] *Keras 2018*. [Online]. Available: https://keras.io/.

[78] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling vnfs using machine learning to improve qos and reduce cost", in *2018 IEEE International Conference on Communications (ICC)*, IEEE, 2018, pp. 1–6.

[79] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data", *arXiv preprint arXiv:1806.00582*, 2018.

[80] "MEC in 5G networks", ETSI, Whitepaper, 2018.

[81] "Feasibility study for Further Advancements for E-UTRA (LTE-Advanced)", 3GPP TR 36.912, 2012.

[82] "5G Frame Structure", Nomor, Whitepaper, 2017.

[83] "UP Latency in NR", 3GPP contribution R2-1711550, 2017.

[84] "5G; NR; User Equipment (UE) radio access capabilities", 3GPP TS 38.306 version 15.3.0 Release 15, 2017.

[85] F. Massimo and M. Ronald, *Random networks for communication: from statistical physics to information systems*. Cambridge University Press, 2008, vol. 24.

[86] A. Othman, S. Y. Ameen, and H. Al-Rizzo, "A new channel quality indicator mapping scheme for high mobility applications in lte systems", *Journal of Modeling and Simulation of Antennas and Propagation*, vol. 1, no. 2, pp. 38–43, 2015.

[87] "Physical layer procedures for data", 3GPP TS 38.214 version 15.3.0 Release 15, 2018.

[88] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[89] D. Harutyunyan, S. Nashid, B. Raouf, and R. Riggio, "Latency–aware service function chain placement in 5g mobile networks", in *IEEE Conference on Network Softwarization*, 2019.

[90] T. Subramanya, G. Baggio, and R. Riggio, "Lightmec: A vendor-agnostic platform for multi-access edge computing", in *2018 14th International Conference on Network and Service Management (CNSM)*, ieee, 2018, pp. 198–204.

[91] *Lightmec*. [Online]. Available: https://github.com/5g-empower/empower-lvap-agent/commits?author=Tejas-Subramanya.

[92] ETSI, "MEC Deployments in 4G and Evolution Towards 5G", Whitepaper, 2018.

[93] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router", *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[94] R. Riggio, S. N. Khan, T. Subramanya, I. G. R. Yahia, D. Lopez, "LightMANO: Converging NFV and SDN at the Edges of the Network", in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*, Taipei, Taiwan, 2018.

[95] *srsLTE*. [Online]. Available: http://www.softwareradiosystems.com/tag/srslte/.

[96] *nextepc*. [Online]. Available: http://nextepc.org/.

[97] *The Squid Caching Proxy*. [Online]. Available: http://www.squid-cache.org/.

[98] *Docker*. [Online]. Available: https://www.docker.com/.

[99] *Metrics-Server*. [Online]. Available: https://github.com/kubernetes-sigs/metrics-server/.

[100] *Prometheus*. [Online]. Available: https://prometheus.io/.

[101] *Grafana*. [Online]. Available: https://grafana.com/.