Deep Learning for Autonomic SLA Management of NFV Resources towards Next Generation Networks

Nikita Jalodia

Department of Computing and Mathematics
School of Science and Computing

South East Technological University

This dissertation is submitted for the degree of

Doctor of Philosophy

Supervisor: Dr. Alan Davy

Submitted to South East Technological University, November 2022

This thesis is dedicated to my *parents— Dr Renu Sharma* and *Rajesh Jalodia*.
Thank you for believing in me, supporting me no matter what, and enabling my independence in this journey.
I owe it all to you.

And of course, to *Mohit*, for being a true partner, and keeping me sane through this.
I could not have done this without you.

That setback was meant to happen.
That rejection was simply redirection.
That failure only pushed you forward.
What you choose to tell yourself becomes your story.

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of Doctor of Philosophy, is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work.

Student ID: 20079552

Signed:  .....

Nikita Jalodia

November 2022

Acknowledgment

This PhD journey has been quite a ride— and I take with me the perseverance and tenacity I've developed in this journey. I have learnt and developed as much personally as professionally, and it would not have been possible without the help and support of the people I've had around in this time.

First and foremost, my utmost gratitude to my supervisor Dr Alan Davy who supported me throughout the ups and downs of this journey, and trusted me at the wheel when the going got tough. I owe it to him for being the enabler of where I stand now. I would also like to thank Science Foundation Ireland (SFI) and CONNECT – the SFI Research Centre for Future Networks and Communications, for funding me and this research, and to Walton Institute for Information and Communication Systems Science for presenting me with the very unique, independent, and yet collaborative research journey.

I would like to thank all the Emerging Networks Lab Research Division Managers— Dr Deirdre Kilbane, Dr Bernard Butler, and Dr Alan Davy. Deirdre, I owe it to you for being so very accommodating the last few months, and for the opportunities you have given me to progress further. Also to Paul Malone for being a great mentor and an even better person, teaching us to smile no matter what cards are dealt – may he rest in peace.

I would also like to thank all the staff in the Research Infrastructure and Testbeds Division, especially John Ronan, Derek O'Keefe, and Michael Kugel for keeping up with me in all my panic modes, often fighting through fire to ensure that the servers and the data-centre stay in the running. It would not have been possible without you, truly.

To Owen and Georgina for making sure the spiders stay far from me in the building, and the many moments of cheerful banter whenever around. To my peers in Walton Institute for their help and support. To WIT Arena for being my stress buster throughout this journey, helping me blow off steam. To Saloni Goyal, for being a great friend and support system, and for introducing me to Headspace in the year that was 2020. I could not have done this without the many meditations that calmed and centered me.

Last but not the least, I am deeply grateful to Dr Mohit Taneja for pushing me out of my comfort zone when needed, and believing in me even when I questioned myself. From the innumerable brainstorming sessions, all our collaborations, and for having my back, always.

Publications

• **[Publication 1 – IEEE ComSoc Open Journal 2021]** N. Jalodia, M. Taneja and A. Davy, "A Deep Neural Network based Multi-Label Classifier for SLA Violation Prediction in a Latency Sensitive NFV Application," in *IEEE Open Journal of the Communications Society*, doi: 10.1109/OJCOMS.2021.3122844. URL: https://ieeexplore.ieee.org/abstract/document/9592636

• **[Publication 2 – IEEE CCNC 2022]** N. Jalodia, M. Taneja, A. Davy and B. Dezfouli, "A Residual LSTM based Multi-Label Classification Framework for Proactive SLA Management in a Latency Critical NFV Application Use-Case," *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022, pp. 782-789, doi: 10.1109/CCNC49033.2022.9700502. URL: https://ieeexplore.ieee.org/abstract/document/9700502

• **[Publication 3 – IEEE Access 2022]** N. Jalodia, M. Taneja and A. Davy, "A Graph Neural Networks based Framework for Topology-Aware Proactive SLA Management in a Latency Critical NFV Application Use-case," revision under review in *IEEE Access*.

• **[Publication 4 – IEEE NFV-SDN 2019]** N. Jalodia, S. Henna and A. Davy, "Deep Reinforcement Learning for Topology-Aware VNF Resource Prediction in NFV Environments," *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2019, pp. 1-5, doi: 10.1109/NFV-SDN47374.2019.9040154. URL: https://ieeexplore.ieee.org/document/9040154

Deep Learning for Autonomic SLA Management of NFV Resources towards Next Generation Networks

Nikita Jalodia

Abstract

Recent advancements in the domain of Network Function Virtualization (NFV), and the rollout of next-generation networks have led to a new era of applications delivered via a paradigm of flexible and softwarized communication networks. This has opened the market to a wider movement towards virtualized applications and services in key verticals such as automated vehicles, smart grid, virtual reality (VR), Internet of Things (IoT), industry 4.0, telecommunications, etc. This has necessitated the requirement for the upkeep of latency-critical application architectures in future networks and communications. While Cloud service providers recognize the evolving mission-critical requirements in latency sensitive verticals, there is a wide gap to bridge the Quality of Service (QoS) constraints for the end-user experience. Most latency-critical services are over-provisioned on all fronts to offer reliability, which is inefficient towards scalability in the long run.

The research presented in this work aims to address the challenges behind effectively managing the trade-off between efficiency and reliability when considering latency critical applications based in a high-availability network slice in next-generation softwarized networks. In the course of research done in this work, we design and develop algorithms to address the complexity towards meeting QoS demands in serving upcoming verticals through the softwarised network architecture, and develop deep learning based frameworks for proactive SLA management in the use-case of a latency-critical NFV application. We utilize data from a real-world deployment to configure and draft a realistic set of Service Level Objectives (SLOs) for a voice based NFV application, and leverage various machine learning based methodologies to proactively identify and predict multiple categories of SLO breaches associated with an application state. With this, we aim to gain granular SLA and SLO violation insights, enabling us to study and mitigate their impact and inform precision in drafting proactive scaling policies in future.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AUC Area Under Curve

AUC-ROC Area Under Curve for the Receiver Operating Characteristic

AUC-PRC Area Under Precision-Recall Curve

ANN Artificial Neural Network

BCE Binary Cross-entropy

BR Binary Relevance

CAPEX Capital Expenditure

CC Classifier Chain

CVIR Coefficient of Variation of the Imbalance Ratio per Label

CNN Convolutional Neural Networks

DNN Deep Neural Networks

DQL Deep Q-Learning

DRL Deep Reinforcement Learning

DDQN Double Deep Q-Network

EB Exabyte

EMR Exact Match Ratio

EDA Exploratory Data Analysis

FN False Negatives

FP False Positives

GRU Gated Recurrent Unit

GConvGRU Graph Convolutional Gated Recurrent Unit Cell

GCN Graph Convolutional Network

GCRN Graph Convolutional Recurrent Network

GNN Graph Neural Network

IR Imbalance Ratio per Label

IO Input/Output

IoT Internet of Things

IP Internet Protocol

IMS IP Multimedia Subsystem

KPI Key Performance Indicator

LSTM Long Short-Term Memory

ML Machine Learning

MANO Management and Orchestration

MDP Markov Decision Process

MAP Maximum a Posteriori

MAE Mean Absolute Error

Mean IR Mean Imbalance Ratio

MSE Mean Squared Error

ML-kNN Multi-label k-nearest neighbors

MLP Multi-Layer Perceptron

NA Network Appliances

NFV Network Function Virtualization

NF Network Functions

NSDs Network Service Descriptors

NN Neural Networks

OPEX Operational Expenditure

PoPs Points of Presence

PCAs Principal Component Analysis

PCs Principal Components

QoE Quality of Experience

QoS Quality of Service

RNN Recurrent Neural Networks

RL Reinforcement Learning

ReLu Rectified Linear Unit

ResNets Residual Networks

SFC Service Function Chain

SLAs Service Level Agreements

SLIs Service Level Indicators

SLOs Service Level Objectives

SDN Software Defined Networking

SVM Support Vector Machine

TN True Negatives

TP True Positives

URLLC Ultra-Reliable Low-Latency Communications

VIMs Virtual Infrastructure Managers

VM Virtual Machine

VNFD Virtual Network Function Descriptors

VNFs Virtual Network Functions

VNFC Virtual Network Functions Component

VR Virtual Reality

VNF-FG VNF Forwarding Graph

VoIP Voice over Internet Protocol

Chapter 1

Introduction

The next generation of networks hold a vision to expand communications from the scale of billions in the world's population to a virtually limitless scale of inter-connectivity between humans, machines, and things. As a result, we are facing a paradigm of exponential growth in enhanced services and applications, network traffic, and consumers. The global mobile traffic is expected to reach 5016 Exabytes (EB) per month in 2030 [1], which is an explosive surge as compared to 51 EB per month in 2020 [2]. Both supporting and driving this demand, the next generation of communication networks continue to be driven by a fundamental restructuring in the way that the networks and services are deployed and delivered. Network programmability and softwarisation are the key drivers of this change, and are delivered via the concepts of Software Defined Networking (SDN) and Network Function Virtualization (NFV) [3]. These continue to play a pivotal role in the vision of 6G, forming the backbone of flexible and intelligent networks [4]. SDN abstracts the underlying network while NFV introduces softwarisation and decouples network functions from the underlying hardware, overall creating a hardware agnostic virtualized environment for network applications [5]. This shift has opened the market to a wider movement towards virtualised applications and services in key verticals such as automatic vehicles, smart grid, virtual reality (VR), internet of things (IoT), industry 4.0, etc., and also includes verticals that previously relied solely on specialised hardware. A key example of such a sector is the telecommunications industry, which is driven by one of the oldest and most complex operational and business support systems to date [4].

Traditionally, with its specialised infrastructure, the telecoms realm has evolved towards a highly reliable service, with carrier-grade offerings guaranteeing a five-nines standard of availability [6]. However, with the emergence of such agile and flexible paradigms as enabled with the coupling of SDN and NFV, we are seeing an emergence of a new era of applications driven by the vision of low latency and high reliability [4]. 5G's usage scenario of ultra-reliable low-latency communications (URLLC) is further expected to

extend in scope to a high-throughput ubiquitous global connectivity at scale, driving all major verticals towards a change [4]. As a result of such a shift, the Cloud infrastructure is no longer host to just web based application services, but is also being extended for the next-generation of requirements that fuel these futuristic application verticals [3]. A key aspect to driving such a change is in how the Cloud reacts to such a latency-critical demand, and in being precisely proactive over time [7].

The remainder of this Chapter presents the research hypothesis that summarises the intent of this dissertation, explicitly defines the scope using research questions, states the research contributions, and outlines the organization of this dissertation.

1.1 Research Hypothesis

The research presented in this work aims to address the complexity towards meeting QoS demands in serving upcoming verticals through the softwarised network architecture. This is directed towards effectively managing the trade-off between efficiency and reliability when considering latency critical applications based in a high-availability network slice in next-generation softwarized networks. To that end, the research hypothesis of this work is as follows:

A combined framework of Graph Neural Networks and Deep Reinforcement Learning can bridge the tradeoff between efficiency and reliability to enable topology-aware dynamic resource management for NFV network services modelled as a multi-component Service Function Chain, and be adept to deal with proactive SLA enforcement for virtual resources.

Building up from the above, we examine four research questions in this dissertation. These are defined as follows:

1.2 Research Questions

1.2.1 First Research Question (RQ1)

Given the use-case scenario of a latency-sensitive NFV based application, what constitutes a suitable SLA definition to track resource provisioning based QoS over time?

This aims to establish and draft a realistic SLA definition that tracks the futuristic complexity of the use-case, which in turn would serve as the basis for optimizing all machine

learning algorithms. This constitutes drafting the SLOs that would constitute the SLA definition, and identifying what service level metrics would be tracked as feature-engineered KPIs input into the learning models. The procedure involves the process of understanding the pains and gains of such a system and use-case in a real-world setup, identifying the bottlenecks that trigger a degradation in QoS for such an application, and using these definitions to label the application data for SLO and SLA violations accordingly. Data engineering and analysis is an inherent part of this phase of study, and involves the transformation of tracked and collected raw system resource monitoring telemetric data into a structured dataset; further followed by cleaning, pre-processing, and appropriate labelling too in this case.

1.2.2 Second Research Question (RQ2)

Given the complexity and dimensionality of data and tracked SLAs, what subset of machine learning solutions are suited to classify the application state at a given point in time as that of an SLA violation for such a task?

This aims to establish baselines in terms of the performance of classical machine learning methods versus deep learning techniques given the complexity of classification methodology required. Key factors include evaluating the processed and labelled data to establish whether the classification problem is balanced or imbalanced, single-output or multi-output, etc. This also involves assessing the level of correlation between output labels, identifying whether they can be treated independently or constitute a more complex overlap that requires non-linear learning methods like deep learning methodologies.

1.2.3 Third Research Question (RQ3)

While tracking the application use-case to proactively avoid SLA violations, do learning methods that track and preserve the inherent topological dependencies perform holistically better than those that work at the higher-level without active knowledge of this metadata? How can this be measured?

This aims to establish the level of granularity in learning that yields the most precise results in proactively forecasting SLA violations over time. While working with time-series data, recurrent neural network architectures like LSTM work at the application level for both single-step and multi-step forecasting, with the choice of both single-output and multi-output predictions towards multivariate models. However, since the NFV application's VNF can be represented as a directed graph of VNFCs through which data flows sequentially, GNN models that also incorporate the node dependencies as topological information

preserved during learning have the capacity to leverage this information towards more precise and effective forecasting. The intent here is to establish the best performing machine learning models in both categories, and compare whether the additional incorporation of spatial metadata enhances the predictive performance over just using temporal information.

1.2.4 Fourth Research Question (RQ4)

How can reinforcement learning enable a dynamic SLA-aware policy enforcement control loop, working towards an adaptable system overseeing the scaling policy at play?

When it comes to dynamic application scenarios, reinforcement learning is an effective machine learning methodology. We can formalize the decision making process here as a markov decision process, which allows us to mathematically represent the environment through states, actions and rewards. Deep reinforcement learning (DRL) uses neural networks as function approximators to deal with a high dimensionality of outcomes while computing their impact over time, and is much better adept towards policy based decision-making for a complex environment with cascading implications. This aims to finish the end-to-end proactive granular control loop. Taking in the forecasted predictions from the model developed through RQ3, the reinforcement learning module is targeted towards dynamic SLA-aware policy enforcement, overseeing the current scaling policy deployed. Given the control over the exploration and reward policy of the reinforcement learning agent, we can potentially aim to optimize the SLA enforcement based on the use-case's target objectives.

1.3 Research Contributions

Summarised on a high-level, the key contributions of this dissertation are:

1. Utilization of data from a real-world deployment to configure and draft a realistic set of SLOs for a voice based NFV application, as presented in Chapter 3.

2. A deep learning based classification strategy to model frequent SLA and SLO violations on the application level as a novel multi-label type multi-output target to enable more complex decision-making in the management of virtualised communication networks. Further, as presented in Chapter 3, a thorough benchmarking of the performance against a set of multi-label compatible machine learning classifiers, followed by a heuristic-based solution to address the challenges in a realistic multi-label setup.

3. A novel multivariate time-series forecasting framework with Residual Long Short-Term Memory (LSTM) based multi-label classification for proactive SLA man-

agement in the latency-critical NFV application use case. Further, as presented in Chapter 4, a benchmarking of the proposed approach against traditional forecasting methodologies to demonstrate its suitability against the state-of-the-art.

4. A novel proactive SLA management framework leveraging spatio-temporal Graph Neural Networks (GNN) and Deep Reinforcement Learning (DRL), leveraging realistic SLA definitions for the use-case to achieve a dynamic SLA-aware oversight for scaling policy management to balance the trade-off between efficiency and reliability. Further, as presented in Chapter 5, a benchmarking of the proposed approach against the previously established best in class model to demonstrate its suitability on the use-case.

1.4 Dissertation Organization

The rest of the dissertation is structured as follows— Chapter 2 presents the related background information that this dissertation builds upon, and a literature review in the areas that the dissertation contributes to. The following three Chapters address the research questions in detail. To this end, Table 1.1 maps the research questions with the contributions

Table 1.1 Research contributions.

Research Question	Research Contributions	Dissertation Chapter	Associated Peer-Reviewed Publication
RQ1 RQ2	• Realistic SLA and SLO definitions for the use-case • Machine/deep learning for predictive classification	Chapter 3	Publication 1 – IEEE ComSoc Open Journal 2021 [8]
RQ3	Traditional (temporal) forecasting on the use-case	Chapter 4	Publication 2 – IEEE CCNC 2022 [9]
RQ3 RQ4	• Graph-based spatio-temporal forecasting • Dynamic SLA enforcement	Chapter 5	Publication 3 – IEEE Access 2022 (under review)
RQ4	Early proof of concept for combining reinforcement learning with traditional graph neural networks (GNN)	Appendix	Publication 4 – IEEE NFV-SDN 2019 [7]

made, and also the corresponding Chapters and associated peer-reviewed publications. Figure 1.1 presents a visual representation of the same, along with the problem domain of each research question, and the areas of contributions.

Chapter 6 presents the concluding remarks for the dissertation, and lists the limitations, and details the directions of future work.

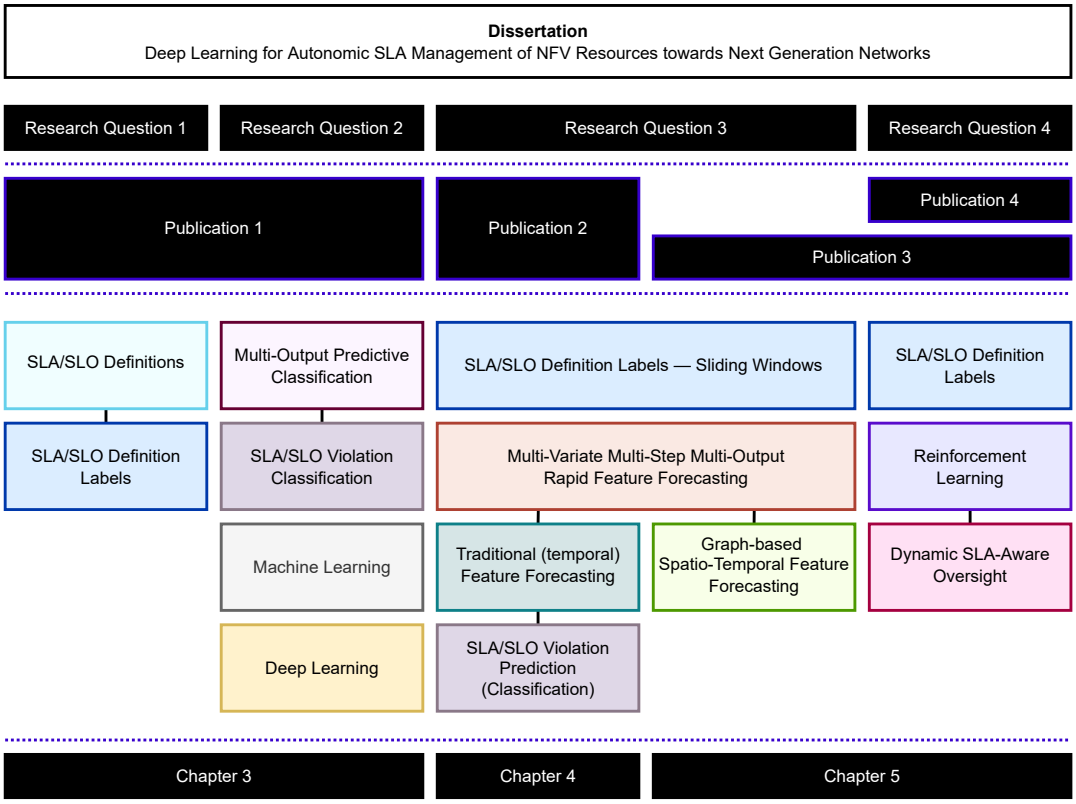


Fig. 1.1 A visual representation of the dissertation work categorized into research questions, peer-reviewed publications, Chapters, contribution areas, and scope.

Chapter 2

Background and Literature Review

In this Chapter, we present the background, motivation, and literature review for the research presented in the dissertation. While the state-of-art is continuously evolving, this study helps in identifying the knowledge gaps that demand further investigation. This Chapter is structured as follows— Section 2.1 to Section 2.3 present the relevant background information and motivation behind the research work. Section 2.4 and its three sub-sections present an in-depth literature review for the areas of contribution of the three main Chapters of the thesis.

2.1 Service Level Agreements (SLAs)

SLAs are closely tied to product and business definitions, and imply a formally explicit consequence upon breach of contract when the agreed terms are violated. While an SLA is a qualitative measure that binds the service provider and facilitator into a formally agreed contract ensuring QoS for the end user, this is realised on a set of low level metrics delivered through SLOs and Service Level Indicators (SLIs). The SLIs can be defined as quantitative measures that build upon raw system metrics, which further feed into the SLOs as a quantitatively definitive target range or threshold towards the deliverance of an SLA. The breach of an SLA implies an explicit consequence, often financial; while the SLOs and SLIs are typically measurable indicators that define the policy of tolerance [10].

While the SLA is a formal contract between a service provider and a consumer, it is often a high-level definition of the service provided. From a service provider's perspective, the SLIs and SLOs are the means to that end, and imperatively define the measurable service characteristics that quantifiably deliver that quality. Therefore, the choice of SLOs are critical towards delivering the QoS promised to the end user, and vary depending on the type of application and use-case scenario.

While the end user Quality of Experience (QoE) may be defined by more than just the server level QoS guarantees, the latter forms the core of the service offered, and is an important characteristic when profiling the service offering.

2.2 Scaling Policies

While static threshold based scaling is still the most dominant scaling policy in use for most systems on the Internet, some Cloud service providers also offer dynamic scaling mechanisms [11]. These have more flexibility on the choice of threshold based on a pre-defined range, and real time network traffic. While such policies offer better adaptation to meeting desired QoS levels, they still do not match the elevated service requirements facing the ebb and flow in network traffic and demand. Acknowledging these shortcomings, some Cloud service providers now also provide an upgrade in the form of predictive scaling policies [12–14]. These are based towards analysing the traffic and key high-level system usage metrics over time, and increasing the system resources during regularly anticipated patterns of high incoming traffic.

While service operators come up with new scaling policies to match the demand facing the current generation of Cloud based application services, these are still a long way to go towards supporting latency-critical applications with high availability values. Since the requirements of real-time applications such as voice and multimedia communication differ from the traditional web-based applications that the Cloud supports, so do the SLAs. A momentary increase in latency and jitter in a voice application has an immediate influence on the end-user QoE, while not quite so in conventional web applications [15]. A key challenge highlighted in the conceptualization of 6G is to impose stringent end-to-end QoS requirements within heterogeneous services [4]. To this end, 5G deployments include the proposition of network slicing, clustering applications with similar demands in an appropriate Cloud environment [5]. This ensures the placement of latency-critical URLLC applications in a high availability network slice, where resources are suitably provisioned to ensure reliability. However, 6G expects an improvement in reliability by at least two orders of magnitude, i.e. from a five-nines standard to a seven-nines standard [4].

Further, efficiency and reliability are competing elements within an SLA, marking a trade-off between system usage and requirements [16]. While network operators and Cloud operators may resort to over-provisioning to match the high requirements for these latency-critical applications in a high availability network slice, such practices are inefficient in the long run [7]. 6G expects a 10 to 100 times improvement on the energy efficiency as compared to 5G [4], which is a challenge in itself. Over time, we need to be mindful of the

significant carbon footprint of Cloud data-centres too, so there is more to it than just the end user experience here [3].

2.3 Quality of Service (QoS) in Softwarized Networks

A transition towards softwarized networks brings in the requirement to adopt more complex models to guarantee QoS and reliability [17], and improve policy-based QoS management [18]. This is because of an impending evolution in not just the way networks are composed and managed, but also renewed application architectures [17, 19], corresponding QoS and SLA management techniques [20], and optimization and automation to cope with the added complexity [21].

Authors in [17, 19] study the impact of virtualization in fault management, and the added challenges that the distinct yet complementary paradigms of SDN and NFV bring in such a setup. Authors in [20] highlight the shift from traditionally tracking the QoS of a single service to that of service compositions in networks, and use a genetic algorithm to optimize the application reliability in the 5G network case. Further, authors in [22] quantitatively model and assess availability from a core network perspective for an end-to-end NFV enabled service. Reliability block diagrams and stochastic reward nets based approaches have also been leveraged for providing an optimal configuration of an NFV based SFC for telecommunications standards availability modeling [6, 23]. An in-depth survey [18] on the autonomic provisioning and QoS management for SDN-based networks highlights the need for more in-depth machine learning models that target and improve policy-based QoS management, and remark that assuring end-user QoE continues to be an open research area.

2.3.1 Forecasting based Resource Management in NFV

Much of the work done so far addresses QoS with characterizing and anticipating traffic patterns, anomaly detection [24], and a combination of reactive and proactive scaling policies [25], [26], [27]. Significant progress has been made in the context of forecasting and clustering anticipated network traffic [28, 29], using machine learning to classify network traffic in NFV [30, 31], and related resource allocation [7, 16]. From a feature forecasting perspective, authors in [32] present an overview of linear and non-linear forecasting methods, and discuss their use to improve multi-slice resource management in 5G networks. Recurrent Neural Network (RNN) based approaches [32–34] have also been successfully applied in the area of communication networks for resource forecasting and management [16].

2.3.2 Automated SLA Management

Post appropriate provisioning following anticipated and identified traffic patterns, there is not a lot of work that directly addresses the remaining SLA bottlenecks from an application perspective. Automated SLA management for use-cases deployed on softwarized networks has been highlighted to be a critical requirement for next generation networks [15, 35–39]. A theoretical SLA management framework that maps high-level requirements to low-level resource attributes is presented in [36], where the authors highlight the additional challenges that 5G and future architectures present. Authors in [37, 38] present a cognitive management architecture for these softwarized networks, and discuss the importance of machine learning techniques in such complete end-to-end management control loops. Existing work on SLA and SLO violation prediction approaches it as a single label output classification [40]— either identifying an overall SLA violation with a binary classification, or identifying a defined SLO breach with multi-class classification [41]. A proof of concept for SLA enforcement in programmable networks in a Cloud-based environment is presented in [42], where the authors work towards identifying an SLO breach with a multi-class decision tree classification methodology.

2.4 Literature Review in Contribution Areas of the Thesis

2.4.1 Multi-Output Classification Models for Automated SLA Management in Latency Sensitive NFV Applications

As mentioned above, post appropriate provisioning following anticipated and identified traffic patterns, there is not a lot of work that directly addresses the remaining SLA bottlenecks from an application perspective. Automated SLA management for use-cases deployed on softwarized networks has been highlighted to be a critical requirement for next generation networks [15, 35]. Further, in a realistic scenario, there is a pressing need for the incorporation of multi-output models as we move towards more complex decision-making [43]. As future networks as well as deployed services gain complexity, it is impractical to define and consider an SLO as a mutually exclusive single-output target. There is no existing work in the area of SLA management that leverages advanced classification methodologies for a multi-output prediction target, identifying and predicting multiple categories of SLO breaches as applicable to study their impact.

To fill this gap, our work as presented in Chapter 3 proposes the use of multi-label classification methodology for a multi-output SLO violation prediction in NFV environments. Multi-label classification is a branch of predictive classification models that involves training models to associate a sample of input data features with more than one class

labels [44]. While the primary motivation for such models draws from the domain of text categorization, image and multimedia object interpretation, music information retrieval, movie genre classification, automated video annotation, etc., other fields such as biology and functional genomics have also leveraged multi-label classification models to address challenging research problems [45]. While initial approaches focused on machine learning based methods to handle multi-label problems [46–49], there has been a recent rise in the application of several neural network architectures to address the complexity and of varied use-cases [43, 50–54]. Associating structured data with multiple semantic information at once holds tremendous potential in the future as we advance towards solving more complex decision making problems [43].

To the best of our knowledge, this has been the first approach in the area that applies a multi-label classification methodology towards a more granular SLA violation prediction for a latency-sensitive VNF in a virtualised network environment, and works with extensive real world data to compare the performance of both machine learning and deep learning methodologies towards such an objective.

2.4.2 Feature Forecasting Methods for Proactive SLA Management in Latency Critical NFV Applications

As mentioned earlier, LSTM based approaches [32–34] have been successfully applied in the area of communication networks for resource forecasting. However, LSTMs are computationally expensive [32]. There has been no comparative study to evaluate the suitability of LSTM-based methods in scenarios that involve rapid forecasting in a realistic high-frequency monitoring, something that is expected to be a critical characteristic for latency sensitive NFV applications. ResNets (Residual Networks) in deep learning refer to architectures where each layer adds to the model's accumulating result [55]. Integrating that with an LSTM methodology leads to a Residual LSTM model, which is expected to overcome the potential limitations of regular LSTM based approaches in use-cases such as ours. However, there have been very limited applications to the use of Residual LSTM in NFV [56], and that too have been in the domain of network slice reconfiguration.

To fill this gap, our work as presented in Chapter 4 has been the first approach in the area that proposes and applies a Residual LSTM based framework for proactive SLA management in rapid forecasting based resource monitoring of latency sensitive NFV applications, and applies multi-label classification towards such target objectives.

2.4.3 Graph-based Spatio-Temporal Forecasting Methods for Proactive SLA Management in Latency Critical NFV Applications

As mentioned above, LSTM based approaches [32–34] have been successfully applied in the area of communication networks for resource forecasting. Our contribution [9] in the area as presented in Chapter 4 benchmarks various deep learning based forecasting methodologies, and proposes a Residual LSTM based framework for proactive SLA management in rapid forecasting based resource monitoring of latency sensitive NFV applications.

However, networks can inherently be represented in a graph based format of nodes and edges, and so can the application structure and flow of data in the softwarized domain of SDN and NFV [7, 16]. In traditional Machine Learning (ML) approaches, these spatial inter-dependencies are removed in the pre-processing stage, and so the inherent effect of these topological dependencies are not a part of the learning and prediction stage [57]. Graph neural networks (GNNs) are a family of neural networks that deal with signals defined over graphs. Modern GNNs are categorized into four groups: recurrent GNNs, convolutional GNNs, graph autoencoders, and spatial–temporal GNNs [58]. Further, in contrast to conventional Machine Learning (ML) approaches, a GNN based approach is capable of producing accurate predictions even when the underlying topology is changed from what the model was trained on [59].

Within the wider domain of networks, GNN has been successfully used in problem areas addressing networking performance and generalisation to larger networks [59], radio resource management [60], etc. Such methodologies have also been successfully used in the area of SDN, addressing connection management [61], energy-efficient VNF deployment [62], spatio-temporal link state prediction [63], detecting and mitigating data plane attacks [64], predicting the optimal path for Service Function Chain (SFC) deployment and packet-level traffic steering [65], etc. Within the domain of NFV, GNN based methodologies have proven successful in problem areas addressing network slicing management [66], VNF deployment prediction [67], VNF resource prediction [16], finding optimal SFC path [68], etc.

Since GNN successfully extracts and models spatial features and topological dependencies, there is an intrinsic potential to use this in conjunction with a reinforcement learning (RL) based approach [7]. Authors in [69] have used graph convolutional network (GCN) based GNN with actor-critic based DRL towards network planning. The combination of GNN and DRL has also been successfully leveraged to tackle power grid management [70]. Within the domain of SDN, message passing GNN with deep Q-learning has been used towards connection management [61], and routing optimization [71]. Authors in [62] have used a GCN based GNN with double deep Q-network (DDQN) towards energy-efficient

VNF deployment. Within the subject area of NFV, graph network based GNN and DRL have been effectively used towards NFV flow migration [72], [73], and optimal VNF placement [74]. GCN based GNN, and DRL have also been leveraged to address the problem areas involving VNF forwarding graph placement [75], and for virtual network embedding [76], [77].

To the best of our knowledge, our work as presented in Chapter 5 has been the first approach in the area that proposes a Graph Convolutional Recurrent Network (GCRN) model for the use-case, and benchmarks its performance against conventional deep learning models that have demonstrated favorable performance on the use-case in the past. We also use realistic SLO definitions to propose an SLA-aware deep Q-learning based DRL model, packaged together in a framework that delivers topology-aware proactive SLA management in a latency-critical NFV application.

Chapter 3

A Deep Neural Network based Multi-Label Classifier for SLA Violation Prediction in a Latency Sensitive NFV Application

3.1 Introduction

As mentioned in Chapters 1 and 2, the next generation of networks hold a vision to expand communications from the scale of billions in the world's population to a virtually limitless scale of inter-connectivity between humans, machines, and things. Both supporting and driving this demand, the next generation of communication networks continue to be driven by a fundamental restructuring in the way that the networks and services are deployed and delivered. Network programmability and softwarisation are the key drivers of this change, and SDN and NFV continues to play a pivotal role in the vision of 6G, forming the backbone of flexible and intelligent networks [4]. This shift has opened the market to a wider movement towards virtualised applications and services in key verticals, including those that previously relied solely on specialised hardware. A key example of such a sector is the telecommunications industry, which is driven by one of the oldest and most complex operational and business support systems to date [4]. Traditionally, with its specialised infrastructure, the telecoms realm has evolved towards a highly reliant service, with carrier-grade offerings guaranteeing a five-nines standard of availability [6]. However, with the emergence of such agile and flexible paradigms as enabled with the coupling of SDN and NFV, we are seeing an emergence of a new era of applications driven by the vision of low latency and high reliability [4]. A key aspect to driving such a change is in

how the Cloud reacts to such a latency-critical demand, and in being precisely proactive over time [7].

While service operators come up with new scaling policies to match the demand facing the current generation of Cloud based application services, these are still a long way to go towards supporting latency-critical applications with high availability values. Since the requirements of real-time applications such as voice and multimedia communication differ from the traditional web-based applications that the Cloud supports, so do the Service Level Agreements (SLAs). A momentary increase in latency and jitter in a voice application has an immediate influence on the end-user Quality of Experience (QoE), while not quite so in conventional web applications [15]. A key challenge highlighted in the conceptualization of 6G is to impose stringent end-to-end QoS requirements within heterogeneous services [4]. Further, efficiency and reliability are competing elements within an SLA, marking a trade-off between system usage and requirements [16]. While network operators and Cloud operators may resort to over-provisioning to match the high requirements for these latency-critical applications in a high availability network slice, such practices are inefficient in the long run [7]. 6G expects a 10 to 100 times improvement on the energy efficiency as compared to 5G [4], which is a challenge in itself. Over time, we need to be mindful of the significant carbon footprint of Cloud data-centres too, so there is more to it than just the end user experience here [3].

In this Chapter, we take the example of a latency-critical NFV application, and draft realistic Service Level Objectives (SLOs) in a way that provide more granular insights into the violations occurring in operational settings. Such insights would help towards improving the formulation of resource provisioning policies in a way that is targeted towards the precise objectives that match service requirements of the target application use-case, rather than a blanket over-provisioning of all categories of system resources. Contrary to simplistic classification methodologies that predict a single label that categorises whether the SLA is violated at the application level, we formulate our work as a multi-label classification problem. This methodology involves training models to associate a sample of input data features with a set of labels from a bigger set of disjoint labels [43, 44], thus helping us to model individual SLO violations associated to the application's state that contribute to an SLA violation overall. We further provide a detailed analysis on how to manage the challenges that such a system presents, including class imbalance with minority classes. We test the performance against a cohort of machine learning solutions, and present a methodical analysis towards the development and effective use of a deep learning classifier for such objectives.

To the best of our knowledge, this is the first approach in the area that applies multi-label classification towards such objectives, and formulates a methodology that combines

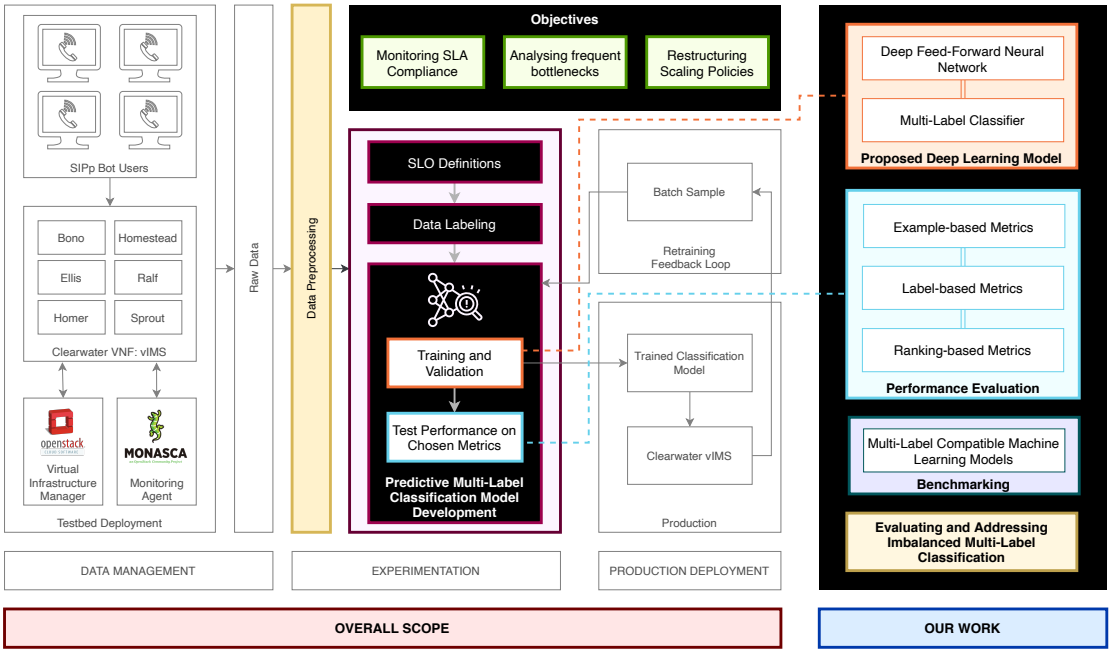


Fig. 3.1 Overview of system architecture, objectives, and scope.

realistic SLO definitions to predict precise QoS violations for such a latency-sensitive use-case. The key contributions are summarised as follows:

- We work with data from a real-world deployment of a latency critical NFV application with two months' worth of raw network telemetry data sampled every 30 seconds, and use that as the basis for all our policy formation and learning models. An overview of the system and scope is provided in Figure 3.1. The data-set is further elaborated upon in 3.6.1.
- We break down the SLA into a set of realistic SLO definitions for such a latency critical use-case in an operational setting. While SLA and SLO definitions are application specific, we form these measures to be as realistic as possible to capture the dynamics of a real-world deployment.
- As opposed to a single-label binary or multi-class classification objective, we associate and model a multi-label classifier to effectively predict each SLO violation that an application state is associated with. Over time, this helps us to study and mitigate frequent application and use-case specific bottlenecks, and also in predicting a more granular state of the application's behaviour as it faces a drop in QoS, and a violation in SLA.
- We test the performance of the developed model against a wide set of compatible machine learning methodologies, and provide a justified reasoning to the deployment of a deep neural network model in such a setup.

- We methodically address the challenges that come up with training such a model at scale, i.e. the associated problems of varying degrees of class imbalance in a multi-label setup.
- We evaluate the performance on a wide range of metrics that include example based, label based and ranking based measures, and provide an all-round evaluation of each learning model benchmarked.

The rest of the Chapter has been structured as follows: §3.3 describes the Clearwater NFV application, and defines the SLA and SLOs drafted for the purpose of violation prediction. §3.4 provides an overview of the unique characteristics of a multi-label classification methodology, the mathematical formulation of the problem statement, the machine learning algorithms applied, and the definitions of the various metrics used for an all-round evaluation. §3.5 addresses the prevailing issue of class imbalance in the multi-label context, and presents the methodologies we use to overcome this issue with a deep neural network model. Thereafter, §3.6 expands on the details of the experimental setup, §3.7 evaluates the results obtained through the various models, and §3.8 presents the summary and conclusion.

The work presented in this Chapter has been disseminated in **[Publication 1 – IEEE ComSoc Open Journal 2021 [8]]**.

3.2 Focus and Scope

This Chapter addresses the first research question (RQ1), i.e.

Given the use-case scenario of a latency-sensitive NFV based application, what constitutes a suitable SLA definition to track resource provisioning based QoS over time?

and the second research question (RQ2), i.e.

Given the complexity and dimensionality of data and tracked SLAs, what subset of machine learning solutions are suited to classify the application state at a given point in time as that of an SLA violation for such a task?

3.3 Defining Service Level Agreements

As mentioned in Chapter 2, SLAs are closely tied to product and business definitions, and imply a formally explicit consequence upon breach of contract when the agreed terms are violated. While an SLA is a qualitative measure that binds the service provider and

facilitator into a formally agreed contract ensuring QoS for the end user, this is realised on a set of low level metrics delivered through SLOs and Service Level Indicators (SLIs). The SLIs can be defined as quantitative measures that build upon raw system metrics, which further feed into the SLOs as a quantitatively definitive target range or threshold towards the deliverance of an SLA. The breach of an SLA implies an explicit consequence, often financial; while the SLOs and SLIs are typically measurable indicators that define the policy of tolerance [10].

While the SLA is a formal contract between a service provider and a consumer, it is often a high-level definition of the service provided. From a service provider's perspective, the SLIs and SLOs are the means to that end, and imperatively define the measurable service characteristics that quantifiably deliver that quality. Therefore, the choice of SLOs are critical towards delivering the QoS promised to the end user, and vary depending on the type of application and use-case scenario.

While the end user QoE may be defined by more than just the server level QoS guarantees, the latter forms the core of the service offered, and is an important characteristic when profiling the service offering.

3.3.1 Project Clearwater Cloud IMS

The IP Multimedia Subsystem (IMS) is a reference architecture first defined by the 3GPP for delivering fixed-line and mobile communications applications built on the Internet Protocol (IP) [78]. Project Clearwater[1] is an open-source implementation of IMS in the Cloud, following IMS architectural principles and supporting all of the key standardized interfaces expected of an IMS core network. The web services-oriented design inherent to Clearwater makes it ideal for instantiation within NFV environments as a virtualized VNF. The new Service-Based Architecture adopted by the 5G standards is very closely related to the inherent Clearwater model, and it has been widely used in research as a standard testbed setup for NFV related work [6, 7, 16, 17, 42].

In our work, we use Clearwater as the use-case for a Cloud based virtualised NFV application. It consists of 6 main components, namely Bono, Ellis, Homer, Homestead, Ralf, and Sprout. A high level view of these VNFCs and their functionalities replicating a standard IMS architecture is as shown in Figure 3.2.

3.3.2 Defining SLOs

We use raw network telemetry data and system metrics obtained via a standard realization of the Clearwater testbed setup to define the SLIs and SLOs governing an SLA. These

[1] www.projectclearwater.org

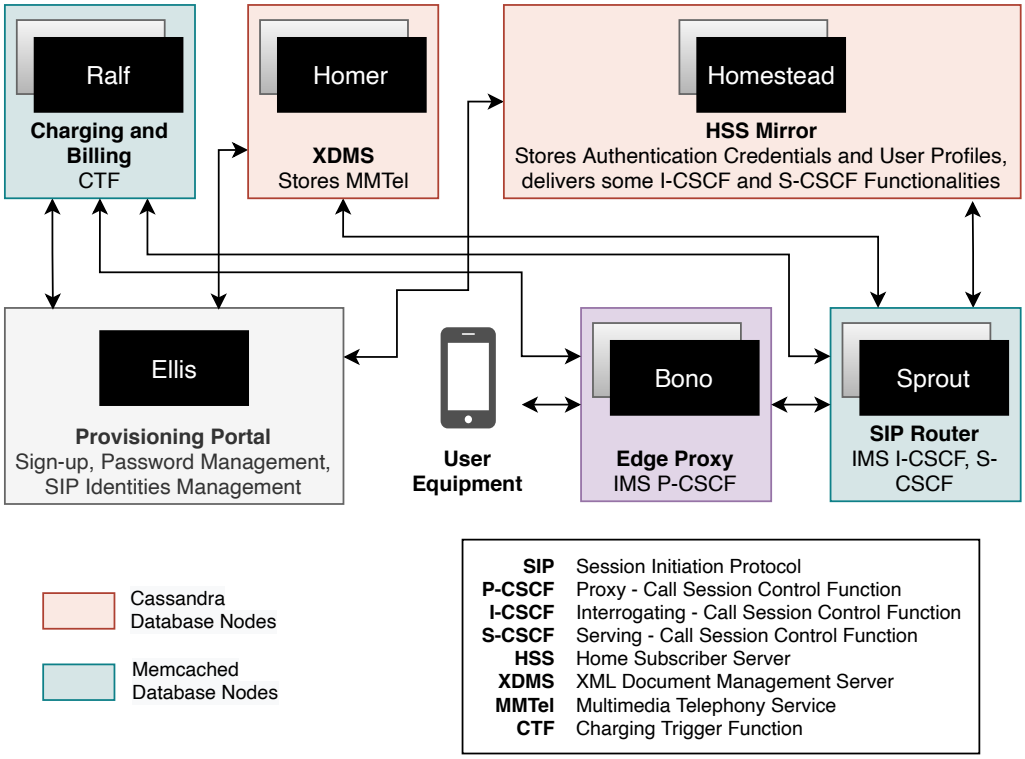


Fig. 3.2 Clearwater vIMS architecture, depicting the various VNFCs and their high-level functionalities.

Table 3.1 Raw metrics collected through Monasca during Clearwater vIMS application monitoring. This data is available for each of the VNFCs, and is sampled every 30 seconds.

		Metric Name	Semantics
CPU		cpu.idle_perc	Percentage of time the CPU is idle when no IO requests are in progress
		cpu.system_perc	Percentage of time the CPU is used at the system level
		cpu.wait_perc	Percentage of time the CPU is idle AND there is at least one IO request in progress
Disk	**Disk**	disk.inode_used_perc	The percentage of inodes that are used on a device
		disk.space_used_perc	The percentage of disk space that is being used on a device
	IO Read	io.read_kbytes_sec	Kbytes/sec read by an IO device
		io.read_req_sec	Number of read requests/sec to an IO device
		io.read_time_sec	Amount of read time in seconds to an IO device
	IO Write	io.write_kbytes_sec	Kbytes/sec written by an IO device
		io.write_req_sec	Number of write requests/sec to an IO device
		io.write_time_sec	Amount of write time in seconds to an IO device
Load		load.avg_1_min	The normalized (by number of logical cores) average system load over a 1 minute period
		load.avg_15_min	The normalized (by number of logical cores) average system load over a 15 minute period
		load.avg_5_min	The normalized (by number of logical cores) average system load over a 5 minute period
Memory		mem.free_mb	Mbytes of free memory
		mem.usable_mb	Total Mbytes of usable memory
		mem.usable_perc	Percentage of total memory that is usable
Network	**In**	net.in_bytes_sec	Number of network bytes received per second
		net.in_packets_sec	Number of network packets received per second
	Out	net.out_bytes_sec	Number of network bytes sent per second
		net.out_packets_sec	Number of network packets sent per second

metrics were collected on a 30 second sampling frequency through Monasca[2], an open-source Python based monitoring service running on each of the Clearwater VNFCs. The list of these collected metrics is presented in Table 3.1, while further details regarding the data is elaborated upon in Section 3.6.

These collected metrics are utilised as the foundations for the SLIs, which when matched with a target threshold or range form SLOs. While the SLOs are largely dependent on the kind of application use-case, and the underlying SLA, we recognize them on the basis of the four key areas that are critical towards the deliverance of required performance. Authors in [42] recognize the lack of realistic SLOs in consideration in research, and recommend that an SLO be composed of a combination of atleast two metrics.

To set a fair ground for our analysis, we define the SLOs with this definition in mind, and form these rules for the four key areas that impact the performance of an underlying system. This is to highlight the varying reason behind the loss of QoS at any time withing the use-case application, so that the scaling policies can be customised at a more granular level towards better efficiency.

Formally, the SLOs are defined in terms of SLIs as a target value:

$$SLI \leq target\ threshold \tag{3.1}$$

or as a range of values for service level:

$$lowerbound \leq SLI \leq upperbound \tag{3.2}$$

At any time, the state of an SLO can be represented as either violated or compliant. We define four SLOs for the Clearwater VNF, targeting the load, computation, disk, and input/output (IO) characteristics respectively. Let $\mathscr{L}$ denote the set of SLOs thus defined:

$$\mathscr{L} = [SLO_1, SLO_2, SLO_3, SLO_4] \tag{3.3}$$

This equivalently denotes:

$$\mathscr{L} = [SLO_{load}, SLO_{computation}, SLO_{disk}, SLO_{io}] \tag{3.4}$$

The metrics as defined in Table 3.1 are captured at the granularity of the individual VNFCs as shown in Figure 3.2, and an SLO violation at any of the individual VNFCs triggers an SLO violation state for the Clearwater application service. Therefore, we ultimately define the SLOs at the application level, i.e. for the entire VNF as an application service. Thus, each data instance is associated with 4 SLOs as defined by $\mathscr{L}$ above, where

[2] www.monasca.io

$SLO_j, j \in [1,2,3,4]$ assumes one of two states:

$$SLO_j = \begin{cases} 1, & \text{if } Violated \text{ (at any VNFC)} \\ 0, & \text{otherwise} \end{cases} \tag{3.5}$$

The SLOs are formulated after studying the scaling policies, dynamic monitoring offerings, and alarm definitions adopted and used in practice by major cloud service providers such as Amazon Web Services [79, 80], Microsoft Azure [81], Google Cloud [82], and Huawei Cloud [83]. We categorise the SLOs into four broad characteristics, and enhance these for a fine-grained monitoring of a latency-sensitive application that needs high availability and reliability. The formal definitions of the SLOs are described below, with the thresholds largely defined based on the application's usage characteristics, reaction to stress tests, and use-case requirements. The metrics referenced in the rule definitions are as captured and described in Table 3.1.

3.3.2.1 SLO_1: Load

Load is a measure of the computational work ongoing, and captures the running processes—either using the CPU, or in a wait state. The values are normalized by number of CPU cores. This SLO captures the application state based on the average load on an instance over a period of the last 1 minute, 5 minutes, and 15 minutes. A short term surge in load may be due to regular operational usage and thus may not be a direct cause of concern, but higher load averages over longer intervals is a direct sign of overload. The data instances that meet the following criterion are assigned a violation state for SLO_1.

$$\begin{gathered} (load.avg_{1min} \geq \gamma_1 \;\; and \;\; load.avg_{5min} \geq \gamma_2) \\ \text{or} \\ (load.avg_{15min} \geq \gamma_3) \end{gathered} \tag{3.6}$$

γ_1, γ_2, and γ_3 are defined as tunable threshold parameters, and were given the respective value of 0.7, 0.5, and 0.8 in the experimentation.

3.3.2.2 SLO_2: Computation

This SLO is defined as a combination of certain CPU and RAM characteristics. While short bursts of IO can spike system kernel usage and is regular, this combined with the lack of adequate idle time for the CPU over time when no IO is in progress is a sign of overload or malfunction. The SLO is also considered violated if the amount of available RAM falls

below a threshold, which is a warning sign of inadequate system resource allocation.

$$\begin{gathered}(mem.usable_{perc} \leq \gamma_4) \\ \text{or} \\ (cpu.system_{perc} \geq \gamma_5 \;\; and \;\; cpu.idle_{perc} \leq \gamma_6)\end{gathered} \tag{3.7}$$

γ_4, γ_5, and γ_6 are defined as tunable threshold parameters, and were given the respective value of 40, 10, and 60 in the experimentation.

3.3.2.3 SLO_3: Disk

This SLO captures prolonged periods of inefficient IO wait times when the CPU is otherwise idle, which indicates potential bottlenecks in the read/write operations accrued by the hard disk.

$$\begin{gathered}(cpu.wait_{perc} \geq \gamma_7) \\ \text{or} \\ \left(\frac{cpu.wait_{perc}}{cpu.system_{perc}} \geq \gamma_8\right)\end{gathered} \tag{3.8}$$

γ_7, and γ_8 are defined as tunable threshold parameters, and were given the respective value of 50, and 2 in the experimentation.

3.3.2.4 SLO_4: IO

This SLO captures the latency when interacting with IO devices, when there is a sudden and prolonged surge in incoming network traffic as compared to the moving average. A moving average (or rolling mean) is defined as the unweighted mean of the previous M data instances sampled, where the selection of M (sliding window) depends on the degree of smoothing desired since increasing the value of M improves the smoothing at the expense of accuracy. Mathematically, rolling mean with a window of size M at a time period t is denoted as follows, where $a_t, a_{t-1}, \cdots$ represent the value at instance $t, t-1, \cdots$ respectively, and so on.

$$\overline{\text{Rolling Mean}}_t^M = \frac{a_t + a_{t-1} + \cdots + a_{M-(t-1)}}{M} \tag{3.9}$$

We choose M to be 2880 (γ_9) for the network traffic characteristics, which, considering that the sampling happens every 30 seconds, corresponds to 24 hours. For the IO read/write characteristics, we use a moving average over the last 3 sampling instances, so $M = 3$ (γ_{10}), which corresponds to the last 1.5 minutes. Thus, this SLO considers both read/write requests per second as compared to the last 90 seconds, as well as the amount of time spent

reading/writing with an IO device as compared to the last 90 seconds.

$$\begin{gathered}\Big(net.in_{bytes\_sec} > \gamma_{11}\ \overline{net.in}^{\gamma_9}_{bytes\_sec} \\ and \\ net.out_{bytes\_sec} > \overline{net.out}^{\gamma_9}_{bytes\_sec}\Big) \\ \text{and} \\ \Big(io.read_{req\_sec} > \gamma_{11}\ \overline{io.read}^{\gamma_{10}}_{req\_sec} \\ or \\ io.write_{req\_sec} > \gamma_{11}\ \overline{io.write}^{\gamma_{10}}_{req\_sec}\Big) \\ \text{and} \\ \Big(io.read_{time\_sec} > \overline{io.read}^{\gamma_{10}}_{time\_sec} \\ or \\ io.write_{time\_sec} > \overline{io.write}^{\gamma_{10}}_{time\_sec}\Big)\end{gathered} \tag{3.10}$$

γ_{11} is defined as a tunable parameter, and was given the value of 1.5 in the experimentation based on the characteristics of the data within the observed period.

3.4 Multi-Label Classification

Multi-label classification is defined as a classification task where each data sample instance can be assigned n labels from a set of $|\mathscr{L}|$ possible label classes as defined in 3.3 and 3.4, where $n \in [0, \mathscr{L}]$, and $|\mathscr{L}| > 1$. Each of the class labels in $\mathscr{L}$ is binary, i.e. either 0 or 1, where 0 denotes a negative occurrence and 1 denotes the positive occurrence.

This implies that $\mathscr{L}$ is a set of binary classes that are not mutually exclusive, and each sample of input data can be assigned multiple such binary classes as applicable.

In our problem definition, $\mathscr{L}$ is the set of all SLO violation classes, where each class can take a value of 0 or 1, signifying compliance and violation states respectively.

Semantically, a multi-label target can be thought of as a set of labels for each sample. Multi-label classification differs from multi-class classification in that the latter applies mutually exclusive labels to a data sample, which is not the case for multi-label problems. The challenge with multi-label classification is the requirement for such classifiers to treat the multiple classes simultaneously, accounting for the correlated behaviour among them.

3.4.1 Mathematical Formulation

Formally, let $\mathscr{D}$ be a multi-label dataset where $\mathscr{X} = \mathbb{R}^d$ is a d-dimensional input instance space of numerical features, and $\mathscr{L} = \{\lambda_1, \lambda_2, \cdots, \lambda_q\}$ a finite output label space of $|\mathscr{L}| = q$ discrete class labels (with values 0 or 1), and $q > 1$.

The task of multi-label learning is to learn a function $f : \mathscr{X} \longrightarrow 2^{\mathscr{L}}$ from the multi-label training set $\mathscr{S}$ with u examples, $\mathscr{S} = \{(\mathbf{x}_i, Y_i) \mid 1 \leq i \leq u\}$. To compare, multi-class classification can be seen as a special case of multi-label classification where $f : \mathscr{X} \longrightarrow \mathscr{L}$, while in binary classification $f : \mathscr{X} \longrightarrow \{0, 1\}$.

For each multi-label example $(\mathbf{x}_i, Y_i)$, $\mathbf{x}_i \in \mathscr{X}$ is a d-dimensional feature vector $(x_{i1}, x_{i2}, \cdots, x_{id})^\top$, and $Y_i \subseteq \mathscr{L}$ is the set of labels associated with $\mathbf{x}_i$. Label associations can also be represented as a q dimensional binary vector $\mathbf{y}_i = (y_{i1}, y_{i2}, \cdots, y_{iq})^\top = \{0, 1\}^q$, where each element is 1 if the label is relevant, and 0 otherwise. By contrast, in single-label (binary or multi-class) learning, $|Y| = 1$.

3.4.2 Multi-Label Learning Methods

Approaches to solve a multi-label classification problem typically belong to two main categories— (a) problem transformation, and (b) algorithm adaptation [44].

3.4.2.1 Problem Transformation Methods

Problem transformation methods aim to transform and decompose the multi-label learning problem into one or many single-label classification tasks, followed by a re-transformation of the outputs into a multi-label representation. The key idea of problem transformation methods is to fit the data to the well-represented set of existing algorithms.

This methodology can be further grouped into three use-case specific categories based on the kind of transformation required— binary relevance, label ranking, and multi-class classification.

3.4.2.2 Algorithm Adaptation Methods

Algorithm adaptation methods, on the other hand, aim to directly tackle the multi-label learning task by adapting or extending the existing classification algorithms to work with multi-label data directly. Unlike problem transformation methods, the key idea of algorithm adaptation is thus to fit or extend an algorithm to work with a multi-label data representation.

3.4.3 Machine Learning Methodologies

Since multi-label classification can be transformed to a binary classification task with the label transformation method as described above, we use two supported methods to transform the multi-label problem to a single-label problem— Binary Relevance (BR), and Classifier Chains (CC). These enable us to compare the performance of some compatible single-label binary classification algorithms when adapted to our multi-label use-case of predicting SLO violation categories.

3.4.3.1 Binary Relevance (BR)

Given $\mathscr{L} = \{\lambda_j \mid j \in [1,q]\}$ as a finite output label space of q discrete class labels as described above, the Binary Relevance method involves treating the j^{th} class independently, i.e. fitting one binary single-label classifier $\mathfrak{B}$ for each class label λ_j. This is akin to a One-vs-Rest strategy with binary classes, where q binary classifiers each treat one of the q label classes independently. In the rest of the Chapter, Binary Relevance and One-vs-Rest is used interchangeably, and One-vs-Rest implies a binary One-vs-Rest strategy.

For a binary learning algorithm $\mathfrak{B}$ embedded in a problem transformation methodology, the worst-case bound training complexity is $\mathscr{O}(q.\mathscr{F}_{\mathfrak{B}}(u,d))$, and the testing complexity is $\mathscr{O}(q.\mathscr{F}'_{\mathfrak{B}}(d))$, where $\mathscr{F}_{\mathfrak{B}}$ denotes the training complexity of the binary classification algorithm $\mathfrak{B}$ embedded in a problem transformation method, and $\mathscr{F}'_{\mathfrak{B}}$ denotes its corresponding testing complexity.

We use logistic regression as a base classifier within the BR methodology, and evaluate its performance against other methods.

Logistic Regression is a linear classification model that uses the logistic (sigmoid) function to take in the input log-odds and output the probability of outcomes for the binary dependent variable. This is interpreted as a binary classification model by establishing a cutoff threshold on the output probabilities to classify the outcome as belonging to one of the two classes.

3.4.3.2 Classifier Chain (CC)

Given $\mathscr{L} = \{\lambda_j \mid j \in [1,q]\}$ as a finite output label space of q discrete class labels as described above, the Classifier Chain method involves linking q binary classifiers ordered randomly along a chain, where the j^{th} classifier tackles the binary relevance problem of label λ_j. However, the feature-space of the j^{th} classifier in CC is extended with the binary label associations of all the previous classifiers linked before it in the chain, thus also exploiting label correlations to an extent.

Classifier chains have a worst-case bound training complexity of $\mathcal{O}(q.\mathcal{F}_{\mathfrak{B}}(u,d+q))$, and a testing complexity of $\mathcal{O}(q.\mathcal{F}'_{\mathfrak{B}}(d+q))$. To evaluate the CC methodology, we use the the following machine learning algorithms as base binary classifiers— Logistic Regression, Naive Bayes, AdaBoost, and Support Vector Machine (SVM).

Naive Bayes is a supervised learning probabilistic classifier that leverages Bayes' theorem with an assumption of conditional independence between each pair of features. Owing to the binary feature space, we use the Bernoulli variant for Naive Bayes as a base classifier.

AdaBoost, or Adaptive Boosting, is an ensemble learning classification technique that builds multiple weak learners on the data and adjusts their weights to improve upon misclassifications as they occur, overall resulting in a boosted classifier.

Support Vector Machine (SVM) is a supervised learning methodology that supports both linear and non-linear classification through kernel functions. An SVM classifier is traditionally non-probabilistic, and we deploy one with a Radial Basis Function (RBF) kernel for a non-linear decision function.

3.4.3.3 Multi-Label k-Nearest Neighbours

Multi-label k-nearest neighbors (ML-kNN) extends the k-nearest neighbors (kNN) algorithm, which is an instance based lazy learning algorithm [45]. It works by identifying the k-nearest neighbours for an example instance in the training set, and utilizes the maximum a posteriori (MAP) rule to make a prediction leveraging the labeling information gained through the neighbours. While ML-kNN reasons the relevance of each label separately [48], it inherits the merits of both lazy learning and Bayesian reasoning. ML-kNN has a worst-case bound training complexity of $\mathcal{O}(u^2d+quk))$, and a testing complexity of $\mathcal{O}(ud+qk))$.

3.4.3.4 Decision Trees

Decision Trees are a non-parametric supervised learning methodology that have been adapted for a multi-label setup by adapting the C4.5 algorithm [47]. The algorithm builds a tree-based model with conditional control statements forming decision rules for classification, and assumes label independence in a multi-label setup [48]. Decision tree models belong to the class of white-box family of algorithms, and the depth of the decision tree is analogous to the complexity of the decision rules. Decision tree based multi-label models have a worst-case bound training complexity of $\mathcal{O}(udq))$, and a testing complexity of $\mathcal{O}(uq))$.

3.4.3.5 Random Forest

A random forest is a decision tree based ensemble learning strategy that works as a meta-estimator and fits a number of decision tree classifiers on various sub-samples of the dataset, leverages this information to control over-fitting. Same as with the adapted decision trees above, random forests belong to the algorithm adaptation method family for handling multi-label classification problems. Likewise, multi-label random forest models have a worst-case bound training complexity of $\mathcal{O}(udq))$, and a testing complexity of $\mathcal{O}(uq))$.

3.4.4 Deep Neural Network

Deep learning is a powerful subset of the machine learning domain, that uses over three layers of interconnected neurons (nodes) to create an artificial neural network (ANN) model. Each neuron within a layer represents a mathematical function comprising of inputs, weights, bias, and threshold; and uses an activation function to transform the outputs to a non-linear space to learn and perform more complex tasks. Thus, subject to the right choice of architecture and parameters for the task at hand, ANNs can be trained to address a wide variety of complex tasks, including that of directly addressing multi-label classification.

To this end, we build a deep multi-layer perceptron (MLP) model, i.e. a fully connected deep feed-forward neural network to natively address the multi-label classification problem at hand and compare its performance against other machine learning methodologies. Specific to the task at hand, we appropriately design the model such that the output layer consists of q neurons, each representing a label λ_j in $\mathscr{L}$, where $\mathscr{L} = \{\lambda_j \mid j \in [1,q]\}$. We use sigmoid as the activation function in the output layer, so the j^{th} neuron in that layer outputs the probabilities in the range $[0,1]$ of the data instance belonging to λ_j. Similar as with logistic regression, this is interpretable as a binary classification by setting a cutoff probability threshold value (set to 0.5) for each class label.

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{3.11}$$

The computational complexity for neural network models can be written as $\mathcal{O}(ru \cdot \mathcal{O}(n^3)))$, where r is the number of iterations, and $\mathcal{O}(n^3)$ is the complexity of the underlying matrix multiplications. The worst-case bound complexity of neural networks is thus $\mathcal{O}(n^5))$. However, this is a very wide overestimate, considering that in practice, modern day neural networks are trained efficiently using stochastic gradient descent, a variety of optimizations and efficient activation functions, over-specification, and regularization [84]. To this end, determining the actual complexity of modern neural networks is an active research area.

Table 3.2 Performance of the various Machine Learning Models as compared to the Base Deep Neural Network Model, evaluated on all presented multi-label classification metrics. The best numeric values corresponding to each metric have been highlighted in bold.

		Binary Relevance	Classifier Chain				ML-kNN	Decision Tree	Random Forest	Deep Feed Forward Neural Network
		Logistic Regression	Logistic Regression	AdaBoost	Naive Bayes	SVM				
Training Loss		NA	NA	NA	NA	NA	NA	NA	NA	0.009
AUC-PRC	Micro Average	0.881	0.902	0.978	0.825	0.857	0.854	0.994	0.980	**1.000**
	Macro Average	0.570	0.632	0.749	0.398	0.477	0.523	0.752	0.745	**0.778**
	Weighted Average	0.880	0.904	0.981	0.832	0.841	0.857	0.996	0.981	**0.998**
AUC-ROC	Micro Average	0.957	0.968	0.994	0.927	0.938	0.940	0.997	0.994	**1.000**
	Macro Average	0.701	0.735	0.857	0.651	0.624	0.657	0.885	0.846	**0.969**
	Weighted Average	0.593	0.634	0.902	0.583	0.526	0.567	0.989	0.901	**1.000**
True Positives (TP)		36211	36886	37976	34220	34582	34961	**38062**	38016	38040
False Positives (FP)		3379	2988	656	4216	3063	3539	130	639	**91**
True Negatives (TN)		100034	100425	102757	99197	100350	99874	103283	102774	**103322**
False Negatives (FN)		2008	1333	243	3999	3637	3258	**157**	203	179
Precision	Micro Average	0.915	0.925	0.983	0.890	0.919	0.908	0.997	0.983	**0.998**
	Macro Average	0.643	0.679	**0.790**	0.582	0.451	0.615	0.779	0.745	0.747
	Weighted Average	0.907	0.922	0.982	0.897	0.831	0.882	**0.996**	0.981	**0.996**
Recall	Micro Average	0.947	0.965	0.994	0.895	0.905	0.915	**0.996**	0.995	**0.996**
	Macro Average	0.631	0.678	0.769	0.529	0.500	0.550	**0.775**	0.748	0.747
	Weighted Average	0.947	0.965	0.994	0.895	0.905	0.915	**0.996**	0.995	**0.996**
F-1 Score	Micro Average	0.931	0.945	0.988	0.893	0.912	0.911	0.996	0.989	**0.997**
	Macro Average	0.633	0.678	0.775	0.436	0.474	0.562	**0.777**	0.747	0.747
	Weighted Average	0.925	0.942	0.988	0.856	0.867	0.894	**0.996**	0.988	**0.996**
Jaccard Similarity Coefficient	Micro Average	0.870	0.895	0.977	0.806	0.838	0.837	0.993	0.978	**0.994**
	Macro Average	0.570	0.631	0.759	0.377	0.451	0.520	**0.763**	0.743	0.744
	Weighted Average	0.875	0.899	0.978	0.815	0.831	0.849	**0.994**	0.978	**0.994**
(Subset) Accuracy (or EMR)		0.856	0.882	0.974	0.793	0.825	0.822	0.991	0.976	**0.993**
Hamming Loss		0.038	0.030	0.006	0.058	0.047	0.048	0.002	0.005	**0.001**
Log Loss		1.837	1.206	0.272	3.731	3.630	2.981	**0.266**	0.279	0.289
Subset Zero-One Loss		0.144	0.117	0.025	0.206	0.174	0.178	0.008	0.023	**0.006**
Coverage		1.236	1.194	1.102	1.364	1.305	1.310	1.092	1.100	**1.080**
Average Precision (Label Ranking)		0.963	0.970	0.993	0.940	0.963	0.953	0.996	0.993	**0.999**
Ranking Loss		0.045	0.034	0.007	0.081	0.060	0.065	0.004	0.006	**0.0002**

The choice of architecture and learning model is further elaborated upon in section 3.6.

The evaluation results for the performance of each of these models for the multi-label classification task at hand are as presented in Table 3.2. Results are also discussed in §3.7.

3.4.5 Metrics

Let $\mathscr{T} = \{(\mathbf{x}_i, Y_i) \mid 1 \leq i \leq p\}$ be the test set with p instances, and $f(\cdot)$ be the learned multi-label classifier. For any unseen instance $\mathbf{x} \in X$, the multi-label classifier $f(\cdot)$ predicts $f(\mathbf{x}) \subseteq \mathscr{L}$ as the set of proper labels for $\mathbf{x}$. Correspondingly, let $Y_i \subseteq \mathscr{L}$ and $Z_i \subseteq \mathscr{L}$ denote the sets of ground-truth and predicted labels for an input instance from the p instances in the test set $\mathscr{T}$.

In traditional single-label classification problems such as the ones belonging to binary classification or multi-class classification, accuracy has been the most common evaluation metric, usually complemented by precision, recall, F-measure, and area under the curve for the receiver operating characteristic (AUC-ROC) [46]. However, multi-label classification

requires a wider and contrasting range of metrics for an overall comparison of performance, given the added freedom, flexibility and complexity in such a setup [44, 47, 85]. These can be grouped as example-based, label-based, and rankings-based [44, 45]. The former two belong to the bipartitions-based evaluation category, which is based on the idea of comparing the predicted relevant labels with the corresponding ground truth labels. The ranking based metrics, on the other hand, offer another perspective to measure the generalization performance of multi-label problems, wherein the most relevant label for an example instance is assigned a value of 1, and so on, with the least relevant label assigned a rank of q.

3.4.5.1 Example-based Evaluation

Example-based evaluation metrics are calculated based on the average differences of the predicted and actual sets of labels over all examples of the test set $\mathscr{T}$.

Exact Match Ratio, also known as Subset Accuracy, computes the fraction of examples for which the predicted set of labels is an exact match with the ground-truth labels. This is defined to be the multi-label equivalent of the traditional accuracy metric; and given that it does not distinguish between partially correct and completely incorrect, tends to be an overly strict measure, especially for a larger label space q. It is formally defined as under, where $\cdot$ denotes an indicator function that returns 1 if $true$, and 0 if $false$. The best performance of this metric is 1.

$$Exact\ Match\ Ratio = \frac{1}{p}\sum_{i=1}^{p} Y_i = Z_i \tag{3.12}$$

Accuracy is defined by micro-averaging the Jaccard Similarity Coefficients across all examples, and is defined as:

$$Accuracy = \frac{1}{p}\sum_{i=1}^{p} \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{3.13}$$

where $|\cdot|$ denotes the cardinality, and the Jaccard Similarity Coefficient for the i^{th} example instance is defined as:

$$Jaccard\ Score = \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{3.14}$$

Precision is defined as the proportion of correctly predicted labels to the total number of predicted labels, averaged over all examples.

$$Precision = \frac{1}{p}\sum_{i=1}^{p} \frac{|Y_i \cap Z_i|}{|Z_i|} \tag{3.15}$$

Recall is defined as the proportion of correctly predicted labels to the total number of actual labels, averaged over all examples.

$$Recall = \frac{1}{p}\sum_{i=1}^{p}\frac{|Y_i \cap Z_i|}{|Y_i|} \tag{3.16}$$

F_β Score is defined as a weighted harmonic mean of precision and recall, whereby β controls the weight of recall in the combined scoring. The case when $\beta = 1$ is referred to as the F_1 score (or balanced F_1 score), which implies that precision and recall are weighted equally in the calculation.

$$F_\beta\ Score = (1+\beta^2)\ \frac{Precision \times Recall}{(\beta^2.\ Precision) + Recall} \tag{3.17}$$

$$F_1\ Score = 2\ \frac{|Y_i \cap Z_i|}{|Y_i| + |Z_i|} \tag{3.18}$$

Subset Zero-One Loss is defined as the fraction of imperfectly classified examples, with the best performance at 0.

$$SubsetZeroOneLoss = 1 - ExactMatchRatio \tag{3.19}$$

Hamming Loss is defined as the fraction of labels predicted incorrectly, and accounts for all misclassifications (prediction errors and omission errors) over total number of label classes over all examples. It is more forgiving in that it penalizes only the individual labels. It is formally defined as under, where Δ stands for the symmetric difference between the two sets, the equivalent of XOR in Boolean logic. The best performance of this metric is 0.

$$Hamming\ Loss = \frac{1}{p}\sum_{i=1}^{p}\ \frac{1}{q}\ |Y_i \Delta Z_i| \tag{3.20}$$

Log Loss, also called the cross-entropy loss, is used to evaluate the probability outputs of a classifier instead of its discrete predictions. As applicable in the multi-label context, the binary variant is defined as under, where $P(\cdot)$ is defined as the corresponding probability estimate, a threshold value of which leads to Z_i.

$$\begin{aligned} Log\ Loss = -\frac{1}{p}\sum_{i=1}^{p}\ \Big[& Y_i \cdot log(P(Y_i)) + \\ & (1-Y_i) \cdot log(1-P(Y_i))\Big] \end{aligned} \tag{3.21}$$

3.4.5.2 Label-based Evaluation

Label-based evaluation metrics are calculated by evaluating the classifier's performance on each class label separately, and then returning the micro or macro averaged value across all class labels.

For the j^{th} class label λ_j, TP_j, FP_j, TN_j, FN_j denote the number of True Positive, False Positive, True Negative, and False Negative test samples from $\mathscr{T}$ with respect to λ_j, where $TP_j + FP_j + TN_j + FN_j = p$.

$$\begin{aligned} TP_j &= \Big|\{\mathbf{x}_i \mid \lambda_j \in Y_i \wedge \lambda_j \in Z_i\}\Big| \\ FP_j &= \Big|\{\mathbf{x}_i \mid \lambda_j \notin Y_i \wedge \lambda_j \in Z_i\}\Big| \\ TN_j &= \Big|\{\mathbf{x}_i \mid \lambda_j \notin Y_i \wedge \lambda_j \notin Z_i\}\Big| \\ FN_j &= \Big|\{\mathbf{x}_i \mid \lambda_j \in Y_i \wedge \lambda_j \notin Z_i\}\Big| \end{aligned} \tag{3.22}$$

Any known evaluation measure applicable to a binary classifier can be adapted to a label-based setup. For any binary evaluation metric $\mathscr{B} \in \{Accuracy, Precision, Recall, F_\beta, \cdots\}$ calculated on the basis of $\mathscr{B}(TP_j, FP_j, TN_j, FN_j)$ for a particular label, the overall label based classification metrics can be obtained by one of the following two averaging methodologies:

Macroaveraging, which implies calculating a metric $\mathscr{B}$ for each class in $\mathscr{L}$, and then averaging over all classes. This can be seen as per-class averaging, and since it gives equal weights to all classes, it is a good methodology to highlight the performance of infrequent classes that are nonetheless important.

$$\mathscr{B}_{macro} = \frac{1}{q}\sum_{j=1}^{q} \mathscr{B}\Big(TP_j, FP_j, TN_j, FN_j\Big) \tag{3.23}$$

Microaveraging, which implies calculating a metric $\mathscr{B}$ globally over all the examples in $\mathscr{T}$ together, aggregating the measure over all classes as a whole. This can be seen as per-example averaging, and tends to be dominated by the performance of the most frequently occurring classes within the example space.

$$\mathscr{B}_{micro} = \mathscr{B}\left(\sum_{j=1}^{q} TP_j, \sum_{j=1}^{q} FP_j, \sum_{j=1}^{q} TN_j, \sum_{j=1}^{q} FN_j\right) \tag{3.24}$$

An example of the binary evaluation metrics $\mathscr{B}$ that the above averaging methodologies can be applied on include:

$$\begin{aligned} Accuracy_{(TP_j,FP_j,TN_j,FN_j)} &= \frac{TP_j + TN_j}{TP_j + FP_j + TN_j + FN_j} \\ Precision_{(TP_j,FP_j,TN_j,FN_j)} &= \frac{TP_j}{TP_j + FP_j} \\ Recall_{(TP_j,FP_j,TN_j,FN_j)} &= \frac{TP_j}{TP_j + FN_j} \end{aligned} \tag{3.25}$$

and so on.

3.4.5.3 Ranking-based Evaluation

Ranking based evaluation metrics compare the predicted ranking of the labels with the ground-truth ranking. The rank predicted by a label ranking method for a label λ is denoted as $\mathscr{R}_i(\lambda)$.

Coverage is defined as an evaluation that calculates an average for how far down the list of ranked labels does the classifier need to go in order to cover all the true labels of an example instance. It is useful in use-cases where it is utmost important to get all true labels predicted, even if that means a few extra false positives [46]. Coverage can be also considered an example-based metric as it is firstly computed for each example, and then averaged across the test set $\mathscr{T}$. The smaller the value of coverage, the better the performance. It is common in implementations to remove the subtraction by 1 in the following equation, so as to be able to extend the metric to handle the degenerate case in which an example instance has no true labels associated with it [86].

$$Coverage = \frac{1}{p}\sum_{i=1}^{p} \max_{\lambda \in Y_i} \mathscr{R}_i(\lambda) - 1 \tag{3.26}$$

Average Precision is defined as the average fraction of labels ranked higher than a particular label $\lambda \in Y_i$, which actually are in Y_i. This can be also considered an example-based metric as it is firstly computed for each example, and then averaged across the test set $\mathscr{T}$. The best value for this evaluation metric is 1, with larger values indicating better performance.

$$\begin{aligned} &Average\ Precision = \\ &\frac{1}{p}\sum_{i=1}^{p}\frac{1}{Y_i}\sum_{\lambda \in Y_i}\frac{\left|\{\lambda' \in Y_i \mid \mathscr{R}_i(\lambda') \leq \mathscr{R}_i(\lambda)\}\right|}{\mathscr{R}_i(\lambda)} \end{aligned} \tag{3.27}$$

Ranking Loss is defined as an evaluation of the average proportion of label pairs that are incorrectly ordered for the example instance, i.e. true labels have a lower score than

false labels. $\overline{Y}$ denotes the complementary set of Y in $\mathscr{L}$, where the goal is that the labels in Y be ranked higher than the labels in $\overline{Y}$. This can be also considered an example-based metric as it is firstly computed for each example, and then averaged across the test set $\mathscr{T}$. The best value of this metric is 0.

$$\begin{aligned} \textit{Ranking Loss} &= \frac{1}{p}\sum_{i=1}^{p}\frac{1}{|Y_i||\overline{Y}_i|}\,|E| \\ |E| &= \left\{(\lambda,\lambda') \mid \mathscr{R}_i(\lambda) > \mathscr{R}_i(\lambda'),\ (\lambda,\lambda') \in Y_i \times \overline{Y}_i\right\} \end{aligned} \tag{3.28}$$

Area Under the Curve (AUC) is defined as either the area under the receiver operating characteristic (AUC-ROC) as illustrated in Figure 3.3, or the area under the precision-recall curve (AUC-PRC). AUC is an intuitive representation of the probability of a randomly selected positive example getting a higher ranking than a randomly selected negative example. The instance-based definition of AUC as described below follows closely from the Wilcoxon-Man-Whitney Statistic [87].

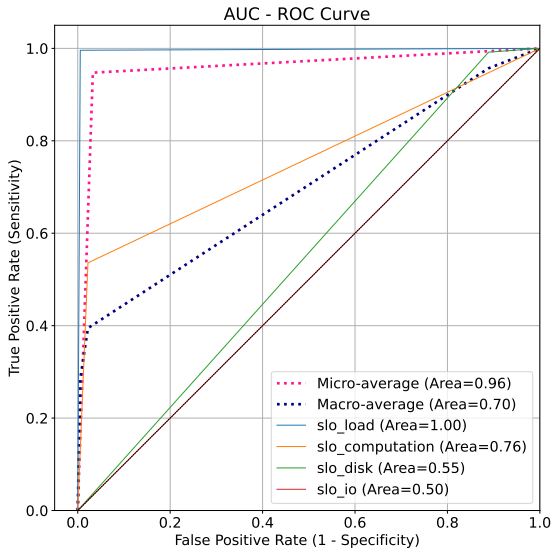
$$AUC = \frac{1}{p}\sum_{i=1}^{p}\frac{\left|\{(\lambda,\lambda') \in Y_i \times \overline{Y}_i \mid \mathscr{R}_i(\lambda') \geq \mathscr{R}_i(\lambda)\}\right|}{|Y_i||\overline{Y}_i|} \tag{3.29}$$

This is a label-based ranking metric, and can be calculated as both a macro and micro averaged value based on Equations 3.23 and 3.24. Its value ranges from 0 to 1, the higher the better.

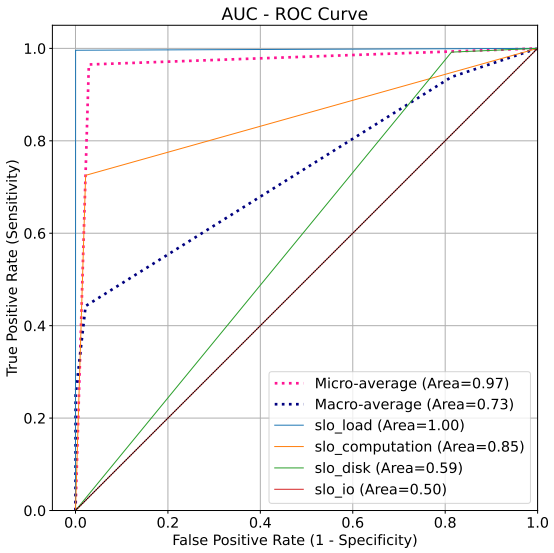
$$AUC\text{-}ROC_j = \int_0^1 TPR_j\, d(FPR_j) \tag{3.30}$$

$$AUC\text{-}PRC_j = \int_0^1 Precision_j\, d(Recall_j) \tag{3.31}$$

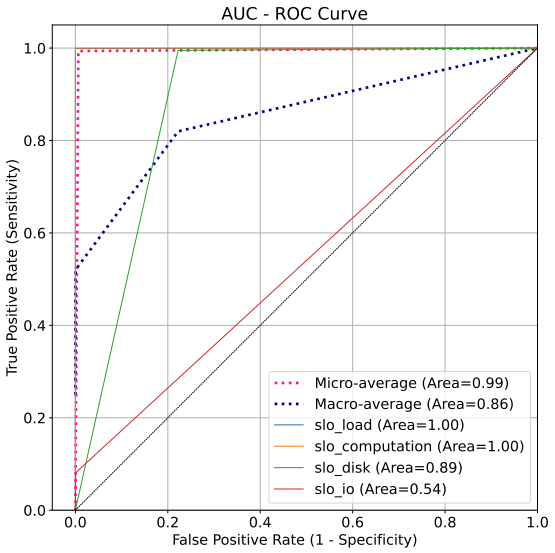
$$\begin{aligned} Sensitivity,\ or\ Recall,\ or\ TPR_j &= \frac{TP_j}{TP_j + FN_j} \\ Specificity,\ or\ TNR_j &= \frac{TN_j}{TN_j + FP_j} \\ FPR_j = 1 - TNR_j &= \frac{FP_j}{TN_j + FP_j} \end{aligned} \tag{3.32}$$



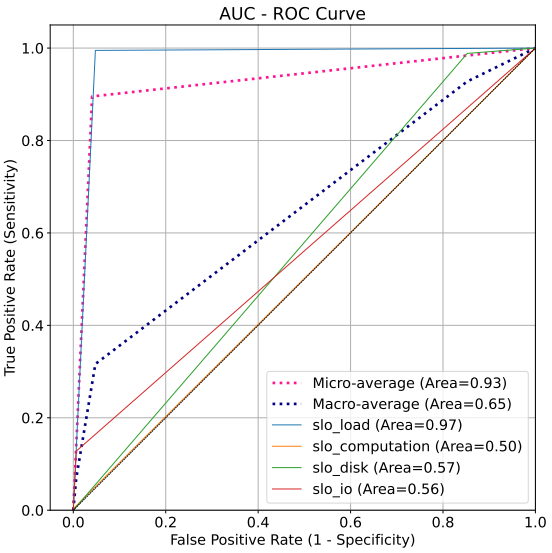
(a) Binary Relevance— Logistic Regression Classifier



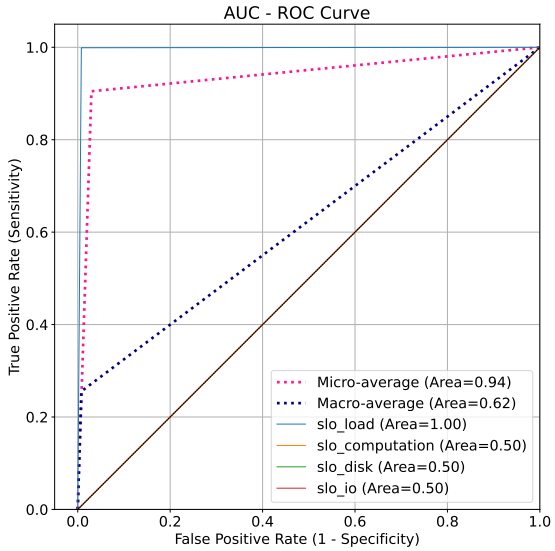
(b) Classifier Chain— Logistic Regression Classifier



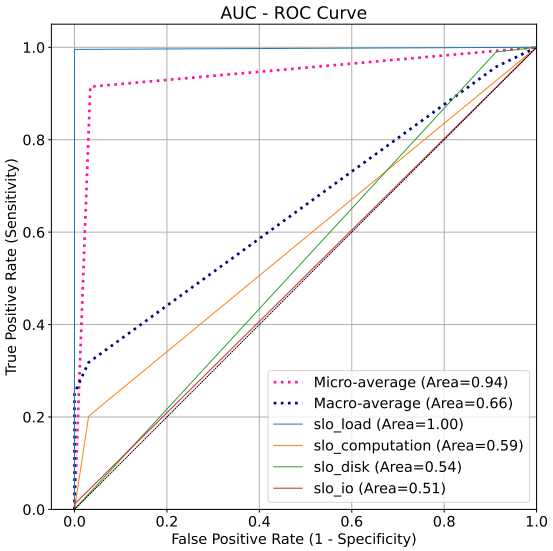
(c) Classifier Chain— AdaBoost Classifier



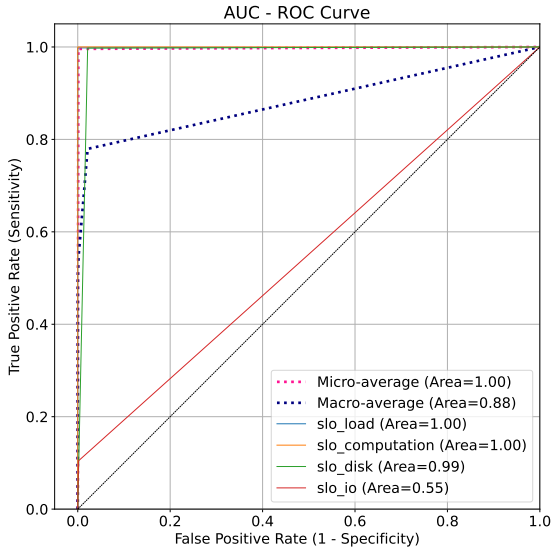
(d) Classifier Chain— Naive Bayes Classifier



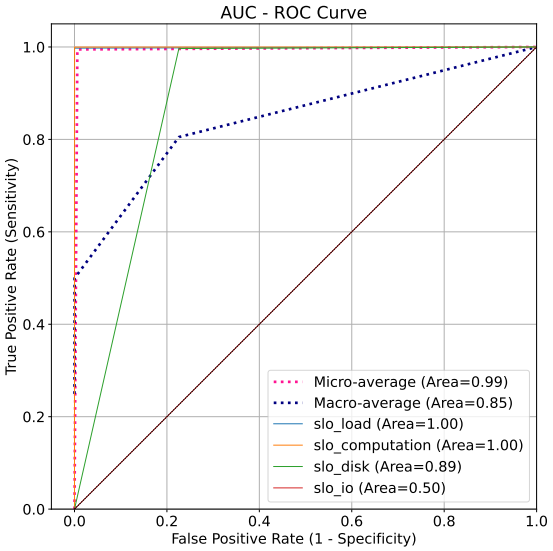
(e) Classifier Chain— Support Vector Machine Classifier



(f) Multi-Label k-Nearest Neighbours Classifier



(g) Decision Tree Classifier



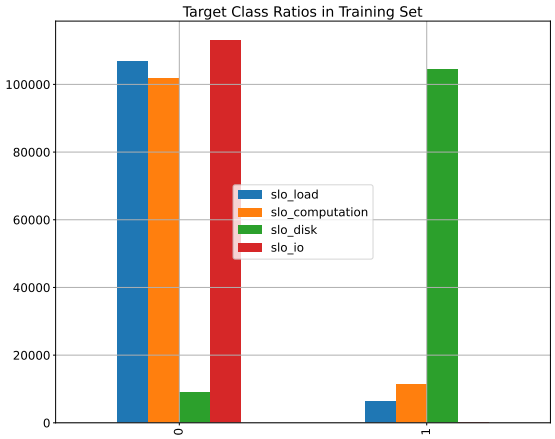
(h) Random Forest Classifier

Fig. 3.3 AUC-ROC plots depicting the learning of the various machine learning classifiers on individual class labels, as well as the micro and macro averaged performance on the entire set.

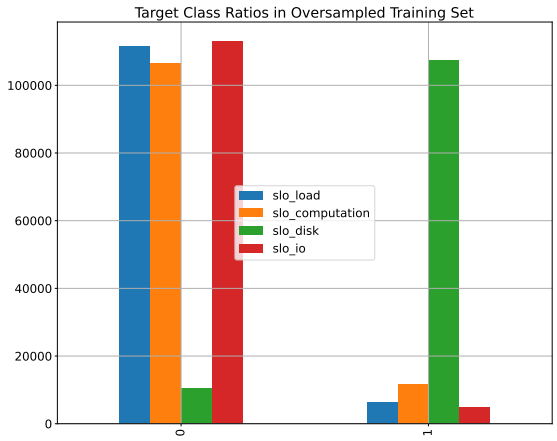
3.5 Addressing Class Imbalance in the Multi-Label Context

Typical classification algorithms perform best when the distribution of data in each of the binary classes is equally distributed. However, when dealing with a high volume of data, especially in the multi-label context, class imbalance is a typical side effect, since not all the labels may be evenly distributed across data instances [43].

(a) Multi-Label Dataset



(b) Training Set



(c) Random Oversampling

Fig. 3.4 Overall distribution of the 4 SLO class labels. The training set contains a positive distribution of 5.64%, 10.05%, 92.16%, 0.23% respectively on the four SLO classes.

Figures 3.4a and 3.4b show a graphical representation of the distribution of the data in each of the SLO classes in our use-case. Owing to the special properties of a multi-label setup, imbalance in a multi-label dataset is measured differently from regular binary or multi-class classification. For the multi-label dataset $\mathscr{D}$ as described earlier, where $\mathscr{D} = \{(\mathbf{x}_i, Y_i) \mid 1 \leq i \leq |\mathscr{D}|\}$, measures such as cardinality and label density are often used in literature to characterize the distribution of labels [44, 48].

Label Cardinality is a measure that defines the average number of class labels associated with each data instance in a multi-label dataset $\mathscr{D}$. It is independent of the number of label classes $|\mathscr{L}| = q$ that exist in $\mathscr{D}$.

$$Cardinality(\mathscr{D}) = \sum_{i=1}^{|\mathscr{D}|} \frac{|Y_i|}{|\mathscr{D}|} \tag{3.33}$$

Label Density is a measure that obtains the ratio of cardinality with the number of label classes that exist in $\mathscr{D}$.

$$Label\ Density(\mathscr{D}) = \frac{Cardinality(\mathscr{D})}{|\mathscr{L}|} \tag{3.34}$$

However, label cardinality and label density do not accurately convey the notion of imbalance [88]. A more concrete measure of the level of imbalance in a multi-label dataset is through the combined use of three specialised metrics— Imbalance Ratio per Label, Mean Imbalance Ratio, and Coefficient of Variation of the Imbalance Ratio per Label [89]. These are defined as follows:

Imbalance Ratio per Label (IR) is a measure that is defined as the ratio between the majority class label λ and each class label $\lambda_j \in \mathscr{L}$. It therefore takes on the value of 1 for the most frequently occurring class label, and a higher value proportionate to the relative degree of imbalance for the other class labels.

$$IR = \frac{argmax_{\lambda \in \mathscr{L}} \left(\sum_{i=1}^{|\mathscr{D}|} \lambda \in Y_i \right)}{\sum_{i=1,\ j=1}^{|\mathscr{D}|,\ |\mathscr{L}|} \lambda_j \in Y_i} \tag{3.35}$$

Mean Imbalance Ratio (Mean IR) is a ratio that presents the mean level of imbalance in $\mathscr{D}$. For example, a Mean IR value of 1.5 represents that there exist, on average, 50% more samples in the majority class label than the minority class label.

$$Mean\ IR = \frac{1}{|\mathscr{L}|} \sum_{j=1}^{|\mathscr{L}|} IR(\lambda_j) \tag{3.36}$$

Coefficient of Variation of the Imbalance Ratio per Label (CVIR) is an indicator of whether all class labels suffer from the same level of imbalance, or if the degree of imbalance differs between them. For example, a CVIR value of 0.2 represents that there exists 20% variance in the IR values among individual class labels. A higher value of

Table 3.3 Metrics capturing the degree of imbalance in the multi-label dataset, and the SLO class labels.

Label Cardinality	Label Density	IR				Mean IR	CVIR
		SLO1	SLO2	SLO3	SLO4		
1.081	0.270	16.259	9.198	1	395.181	105.409	1.833

CVIR implies a higher degree of variation of imbalance between individual class labels.

$$CVIR = \frac{1}{Mean\ IR} \sqrt{\sum_{j=1}^{|\mathscr{L}|} \frac{(IR(\lambda_j) - Mean\ IR)^2}{|\mathscr{L}| - 1}} \tag{3.37}$$

Table 3.3 presents the measure for the degree of imbalance in our use-case data, using the above measures. To compare, a multi-label dataset is considered imbalanced if the Mean IR is higher than 1.5, and CVIR is over 0.2 [89].

There exists class imbalance in all the class labels, but the distribution is quite extremely skewed in the last class label, i.e. SLO_{io}. Such a distribution skews the performance as portrayed by the evaluation metrics, as the classifier may not learn the representations of the minority classes in the training set $\mathscr{S}$. For example, in the cases such as our use-case where the binary distribution of classes between the class labels is skewed to such varying range of extremes, a classifier can achieve 94.12% accuracy by just predicting the majority classes for all the class labels in $\mathscr{L}$ in the test set at all times. This is also the reason why training based on maximising accuracy and other conventional metrics do not perform well in an imbalanced multi-label context.

Moreover, since the prediction value for each neuron in the output layer is a continuous value in $[0,1]$ which is translated to a binary classification by setting a threshold, this creates a trade-off between precision and recall. The AUC-ROC is a great way to ascertain the quality of a predictor without the threshold, and is a very useful metric to evaluate a classifier, especially in the context of an imbalanced class distribution [87]. However, it is much more appropriate to train by aiming to maximize the area under the precision recall curve (AUC-PRC) during training [90, 91].

Post training, while it is helpful to consider the macroaveraged metrics to understand the performance in a multi-label context, they may not independently convey the full picture on the model and its learning capabilities for the individual classes. It may so occur that a class is entirely ignored by the classifier, and its interactions never picked up and modeled in the learning phase, while the performance continues to improve on the macro and microaveraged metrics due to improvements in learning the other classes. As such, for such extreme distributions, it is acutely important to also track each of the class labels individually to understand if the classifier has been modeling them in the learning

phase, and to ascertain if it is performing better than a random classifier (i.e. a classifier that assigns classes randomly). We present a visual representation for the performance of each of the learning classifiers on the individual classes by way of confusion matrices and AUC (both AUC-ROC and AUC-PRC). This aspect adds on to the requirement of tracking the classifier's performance on a wider set of metrics as mentioned earlier, as it helps to better understand the shortfalls of a learning strategy, and reveals a bigger picture behind an overly optimistic prediction performance.

As can be understood from Table 3.2 and elaborated upon in §3.7, neural networks have a direct advantage in multi-label classification algorithms by concurrently understanding the correlations and working on learning all label classes simultaneously. However, neural network architectures do not implicitly support imbalanced classification [92]. There are two ways to directly address class imbalance in classification problems— either to take steps to make models resilient to class imbalance, or to apply rebalancing and resampling techniques to reduce the imbalance in data [93]. Improving multi-label classification for real world data is currently an active research area [94, 95], and so is addressing imbalance in a multi-label setup [96–98]. We address imbalance in our use-case by adapting two conventional methodologies to a multi-label context, and use them to improve the performance of the deep feed-forward neural network model for the minority class labels.

3.5.1 Adding Class Weights

The most common strategy to address class imbalance is to introduce appropriate weights for the minority samples, so as to have a weighted learning cost-sensitive strategy[92]. In single output classification models, this is usually done by weighing each class inversely to the ratio of minority to majority labels within it [99].

3.5.1.1 Deep FFNN with Balanced Class Weights

Since multi-label classification consists of multiple class-labels associated with a data instance, and each of the labels is binary, we adapt the above strategy by weighing each class label in inverse proportion to its frequency of positive to negative occurrence. Thus, for each class label λ_j in $\mathscr{L}$, where $\mathscr{L} = \{\lambda_j \mid j \in [1,q]\}$, we weigh λ_j as per the ratio of its $0:1$ label distribution in the training set $\mathscr{S}$:

$$\frac{|\,Total_j\,|}{|\,Positive_j\,|} - 1 \tag{3.38}$$

which is equivalent to

$$\frac{|\,Negative_j\,|}{|\,Positive_j\,|} \tag{3.39}$$

This ensures that the class labels that have a lower frequency of positive occurrence in the training set $\mathscr{S}$ are weighted relatively higher than the other class labels, and thus a misclassification for these infrequent classes is compensated by penalizing it higher in that proportion. Thus, the classifier is forced to adequately tune the learning model by also considering the behaviour of the minority classes.

So, for a training set $\mathscr{S}$ with 113,303 data instances with a positive distribution of 5.64%, 10.05%, 92.16%, 0.23% respectively on the four SLO classes, we correspondingly apply the relative positive class label weights of 16.74, 8.94, 0.08, 443.32, with this methodology.

3.5.1.2 Deep FFNN with Clipped Class Weights

Neural networks do not train well with large weights [100]. While imbalance in classes is measured by orders of magnitude, the extreme weight value of 443.32 for the last class with the above method, although formally correct, is anticipated to be detrimental for performance. Furthermore, SLO_3, the most positively distributed class label, assumed a weight of 0.08 by the above methodology. This is lower than the relative weight of 1.0 that is assigned to the distribution of all negative occurrences, which should deteriorate the learning for that class label with a drastic addition in the false predictions for that class label.

Thus, in order to test the above hypothesis, we add a model with clipped class weights. Here, we clip the class weights derived from the above methodology such that the maximum value that any of the class labels assume is set to a tunable upper limit (we set it to 100.0), and the minimum value for any weight is 1.0, i.e. equal to that of the negative distributions. Thus, with this methodology on the four SLO classes, we correspondingly apply the class label weights of 16.74, 8.94, 1.0, 100.0.

3.5.1.3 Deep FFNN with Log Smoothed Class Weights

While the above methodology is expected to be an improvement on the first one, setting the upper threshold for class weights is a heuristic nonetheless, and finding its optimal value would require another grid search each time the model is changed. Thus keeping in mind that the neural network weights should be set to low values for optimal training and bias-variance trade-off, we introduce a log smoothed model that combines the benefits of the above two methods.

$$log\left(\alpha \cdot \left[\frac{|\,Total_j\,|}{|\,Positive_j\,|} - 1\right]\right) \tag{3.40}$$

α here is a tunable hyperparameter that can be varied to change how many class labels should be weighed above 1. This depends on the distribution of the positive and negative occurrences within a class label, and the unweighted model's learning performance on the skewed classes. We set α to 0.16, which is its maximum value until which the first three classes are weighed at the default value of 1.0, and only the most skewed class, i.e. SLO_4 takes on a class weight. This is the methodology that seeks to set the weighting to a bare minimum, both on the number of classes as well as the relative ratios. Thus, with this methodology on the four SLO classes, we correspondingly apply the class label weights of 1.0, 1.0, 1.0, 4.26.

3.5.2 Random Oversampling

An alternative to class weighting is the use of oversampling or undersampling techniques to either increase the occurrence of minority class labels, or decrease the sampling of the majority class labels respectively during the training phase [101]. It is to be noted that in the multi-label context, this would also cause indirect over or under sampling the other class labels that co-occur in Y_i on these training instances. Further, the degree of oversampling has a proportionately high likelihood of disrupting the learning model in a multi-label context, and is expected to cause overfitting nonetheless, since the model would see certain data instances more than once during training [96]. Since oversampling increases the size of the training set $\mathscr{S}$, it also affects the distributions of labels overall in that set. While traditional resampling strategies aim to create a balanced dataset from an imbalanced one [101], doing so is not as straightforward in a multi-label context due to the high level of concurrence between imbalanced labels, its translational impact on a large number of unique label-sets, and the resultant introduction of a rather high level of noise in training [96].

To this end, a simplistic methodology involves oversampling one or more class labels until their IR matches up to the Mean IR, or to oversample one or more imbalanced class labels until the size of the training set $|\mathscr{S}|$ is a certain percentage larger than the original [89]. However, the exact dynamics vary largely by the individual data and distribution characteristics, as well as the complexity and performance of the classification algorithm in use. Since a deep neural network model is extremely prone to overfitting and was seen to already perform well in capturing the imbalance in the first three class labels, we adopted a random oversampling strategy wherein we oversample the training set $\mathscr{S}$ during training for the data instances that feature the class label with the most extreme IR

compared to the other class labels, i.e. SLO_4. Corresponding to the IR values in Table 3.3 stating the degree of imbalance in each class label, the actual distribution for each class label (positive occurrence) is 5.64%, 10.05%, 92.16%, and 0.23%. We oversample the instances containing SLO_4 by approximately 4% of $|\mathscr{S}|$, such that the incidence of SLO_4 increases from 0.23% to 4.24%, but still remains below the distribution of the next closest imbalanced class. The new distribution on the bigger training set becomes 5.45%, 9.84%, 91.04%, and 4.24%, with a corresponding IR of 16.70, 9.26, 1, and 21.49. The Mean IR for the new oversampled training set is 12.11, with a CVIR value of 0.74.

With this methodology, the model would see an increased incidence of the most infrequent minority class label in the mini-batches during training, and this is aimed towards increasing the odds that the model will at least be able to capture its behaviour as well as interactions with the other class labels. Nevertheless, the validation and test set remain unchanged, and the training can be readjusted by breaking up the epochs, applying appropriate weight regularization, and providing finer control to early stopping callbacks. Figure 3.4c shows the resulting distribution after we apply the oversampling strategy on our use-case.

Algorithm 3.1: Decision-making strategies towards addressing extreme class label imbalance for SLA violation prediction in a latency sensitive NFV application

Input : Training Data $\mathscr{S} \in \mathbb{R}^d$

Compute IR for each SLO class label $SLO_j \mid j \in [1, |\mathscr{L}|]$ using Eq. 3.35, the Mean IR using Eq. 3.36, and the CVIR using Eq. 3.37

• **CLASS WEIGHTED STRATEGY**

Function `Class Weighted Strategy(`$\mathscr{S}$`):`
- **if** *(Mean IR* > 0.5 *and CVIR* > 0.2*)* **then**
  - Compute class weights using Eq. 3.38
  - **if** *(Class weight of any* $SLO_j > 100$ *or* < 1*)* **then**
    - Compute class weights using Eq. 3.40

End Function

• **RESAMPLING STRATEGY**

Function `Resampling Strategy(`$\mathscr{S}$`):`
- **if** *(IR of* $SLO_j \ggg$ *Mean IR)* **then**
  - Randomly oversample $\mathscr{S}$ for data instances belonging to SLO_j
- **else**
  - Randomly oversample $\mathscr{S}$ until IR for SLO_j gains better proximity to the Mean IR

End Function

Algorithms 3.1 and 3.2 present the consolidated proposition towards effectively addressing class imbalance in a multi-label setup in a deep FFNN model, that extracts the

Algorithm 3.2: A deep feed-forward neural network based multi-label classifier for SLA violation prediction in a latency sensitive NFV application

• **PRE-PROCESSING**
Input : Data: $\mathscr{D} \in \mathbb{R}^d$
Output : Pre-processed training set $\mathscr{S}$, validation set $\mathscr{V}$ and test set $\mathscr{T}$

Function `Pre-Processing(`$\mathscr{D}$`):`
    Split the data in train, test, and validation sets
    Standardize according to the training set $\mathscr{S}$
End Function

• **DEEP FFNN MODEL**
Input: Pre-processed training set $\mathscr{S}$, validation set $\mathscr{V}$ and test set $\mathscr{T}$
Output: Multi-label classification for SLO violations
Function `Deep FFNN Model(`$\mathscr{D}$`):`
    Construct neural network architecture
    if *(Addressing label imbalance with class weight based models)* **then**
        CLASS WEIGHTED STRATEGY
    else
        if *(Addressing label imbalance with random oversampling)* **then**
            RESAMPLING STRATEGY
    Apply appropriate values for all key structural and non-structural hyperparameters
    Specify the Early Stopping criterion
    Compile the deep FFNN model
    repeat
        Train and fit the model with batches from $\mathscr{S}$, using $\mathscr{V}$ as validation set
        Output multi-label classification prediction values
        Monitor and evaluate training with validation set
    until CONVERGENCE;
End Function

Use data from the test set $\mathscr{T}$ to evaluate the trained model

learning from the above four methodologies adapted for our use-case scenario. In the following sections, we present and compare these strategies, highlight model performance and evaluation, and present the challenges through such a setup.

3.6 Experimental Setup

The experiments were all set up using Python (version 3.8.5) and its associated data-science libraries. We use the Scikit-Learn [86] library for all machine learning based methodologies used in this Chapter, and Tensorflow [102] version 2.4.1 with Keras [103] to program all the deep learning based implementations.

3.6.1 Dataset

We use a publicly hosted dataset[3] obtained via a standard Clearwater testbed, a visualization for which is presented in Figure 3.1. The dataset comprises of raw telemetric data files that track 26 metrics for each of the 6 monitored VNFs that compose the Clearwater ecosystem, and includes bursts of abnormal behaviour through its integrated stress testing tools to simulate VNF congestions and QoS degradations. The data is sampled every 30 seconds, and spans an overall period of 2 months. This corresponds to 156 features overall, and 177,098 rows of raw temporal data. A brief description of the captured metrics is summarised in Table 3.1. Outcomes of the exploratory data analysis as performed on this data-set have been presented in Appendix A.

3.6.2 System Configuration

The experiments were performed in a Docker[4] based containerized environment running atop a bare-metal Linux server with 64 GB RAM, Intel® Xeon® CPU E5-2660 v2 @ 2.20GHz (40 physical processors), 2 NVIDIA® Tesla K20m GPUs, and 500 GB local storage. The Docker image runs an Ubuntu 20.04 LTS operating system, and CUDA version 11.3 for the GPUs.

3.6.3 Learning and Adaptation

The data input into any model, be it machine learning or deep learning, is first standardized via mean centering and ensuring a standard deviation of 1, i.e. subtracting each data feature value by its corresponding mean in the training set, and dividing by the standard

[3] www./bit.ly/3gPY8c5
[4] www.docker.com

Table 3.4 Key results from random search for finding an optimal neural network architecture. Best values have been highlighted in bold.

Results of Random Search for Neural Network Architecture											
Neural Network Layers, and Number of Neurons in Each			Loss	TP	FP	TN	FN	Accuracy	Precision	Recall	AUC
Input Layer	Hidden Layers	Output Layer									
126	65, 0.2 (D)	4	0.015	37973	366	103047	246	0.995	0.990	0.993	**0.999**
	65, 0.2 (D), 24		0.010	38043	139	103274	176	**0.997**	**0.996**	**0.995**	**0.999**
	65, 0.2 (D), 65		0.014	37960	248	103165	259	0.996	0.993	0.993	**0.999**
	65, 0.2 (D), 24, 0.1 (D)		0.011	38006	130	103283	213	**0.997**	**0.996**	0.994	**0.999**
	65, 0.2 (D), 65, 0.1 (D), 24, 0.1 (D), 24		0.010	38041	161	103252	178	**0.997**	0.995	**0.995**	**0.999**
	65, 0.3 (D), 65, 0.2 (D), 24, 0.1 (D), 24		**0.008**	38035	**118**	**103295**	184	**0.997**	**0.996**	**0.995**	**0.999**
	65, 0.3 (D), 65, 0.2 (D), 24, 0.2 (D), 24		0.010	**38050**	155	103258	**169**	**0.997**	0.995	**0.995**	**0.999**

deviation. This pre-processing, as also presented in Algorithm 3.2, helps in the learning and convergence of any predictive modeling algorithm [100].

We split the available data into training and test sets in the ratio of 80 : 20, and the training set is further split into training and validation sets in the ratio of 80 : 20. Hence, overall, the data consisting of 177,098 rows is split in the ratio 64 : 20 : 16, corresponding to train, test, and validation splits respectively.

The choice of architecture has an important role to play in the model learning and performance for a neural network methodology. While there are no fixed guidelines on the number of layers and the number of neurons in each of them, the choice is often driven by following a heuristic based on number of inputs and outputs, and using a random search methodology to arrive at an optimal architecture for the use-case and data at hand. We begin with a shallow universal approximation architecture [16], i.e. with one hidden layer, and the number of neurons in it equal to the average of the neurons in the input and output layers. We use its results as a baseline to adjust the number of layers and neurons, and also adjust the dropout regularization factor between layers to control overfitting [104] as the model gains complexity. This results in a deep learning architecture, and the key results from this random search based architecture optimization are presented in Table 3.4.

For FFNN model training, we use Binary Crossentropy as the loss function to be minimized. It is a probabilistic loss function that computes the cross-entropy loss between true and predicted labels, and is appropriate for use in a binary classification based setup. Further, we use Adam [105] as the optimizer for its computational efficiency and adaptive learning, with its default learning rate of 10^{-3}. ReLu (Rectified Linear Unit) is used as the activation function for each of the hidden layers due to its computational simplicity and high optimization performance in a deep MLP based setup [100]. As mentioned earlier, we use sigmoid as the activation function for the output layer to concurrently output the individual probability of each label's association with the input data instance, thus supporting multi-label classification outputs.

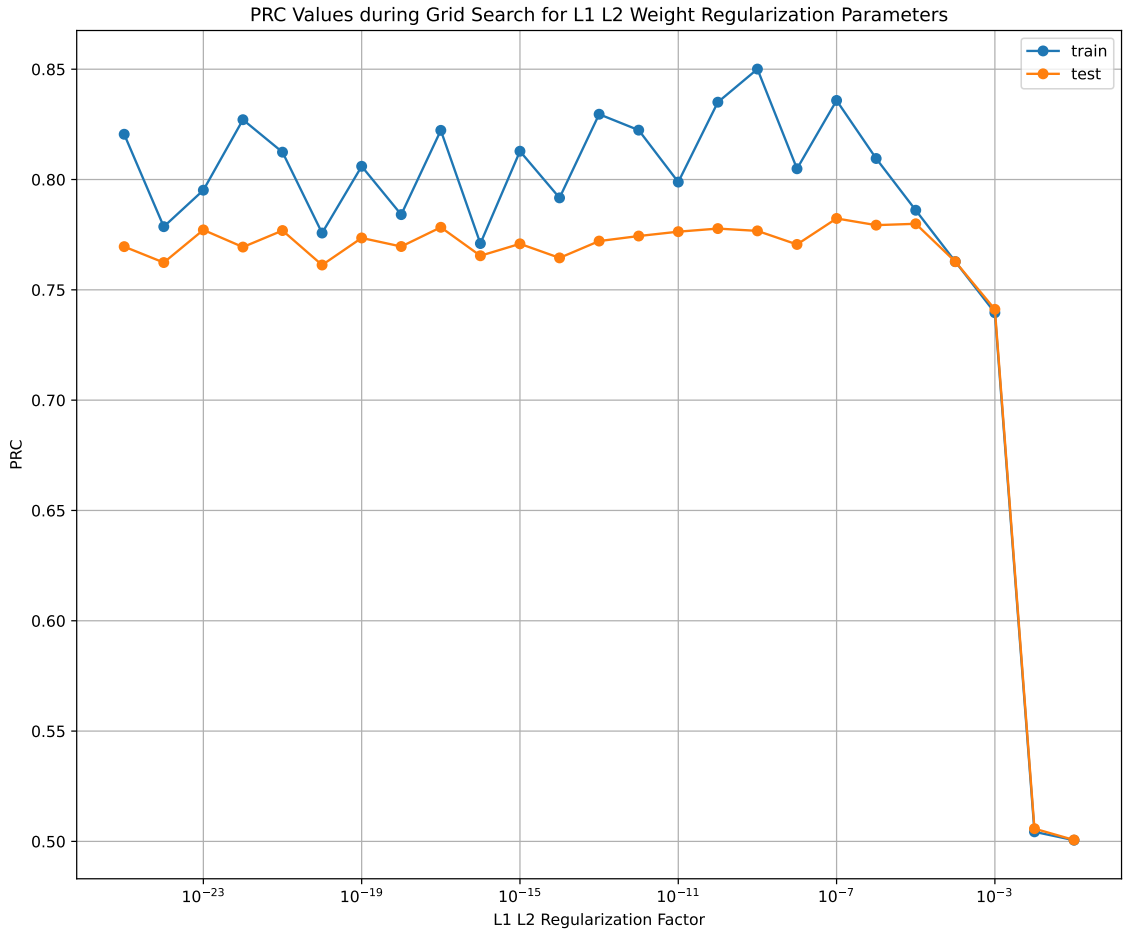


Fig. 3.5 Grid search for L1 L2 weight regularization on the deep FFNN architecture.

Since the label classes are imbalanced, we initialize the bias in the output layer to reflect this and enable the model to begin with more reasonable initial guesses, thus contributing to faster convergence. The bias initialization is derived through the log of the ratio of $positive : negative$ in each of the class labels, averaged over all class labels in the training set. This also eliminates the erratic initial behaviour in the learning loss curve in the initial epochs of training, where the model is just learning the bias.

To further control the degree of overfitting during training, we perform a grid search for the optimal choice of weight regularization hyperparameters for the neural network model, and based on the results, apply both L1 and L2 weight regularization on each of the hidden layers in the FFNN, with a regularization factor of 10^{-7} for the unweighted and class weighted models, and 10^{-4} for the model with random oversampling. A visual representation of the outcome of grid search to this end is presented in Figures 3.5 and 3.6.

We use a large batch size of 2048 to increase the probability of class representation from the minority class labels during the training phase. While we set the maximum training epochs to 500, we also deploy an early stopping criteria that tracks the macroaveraged

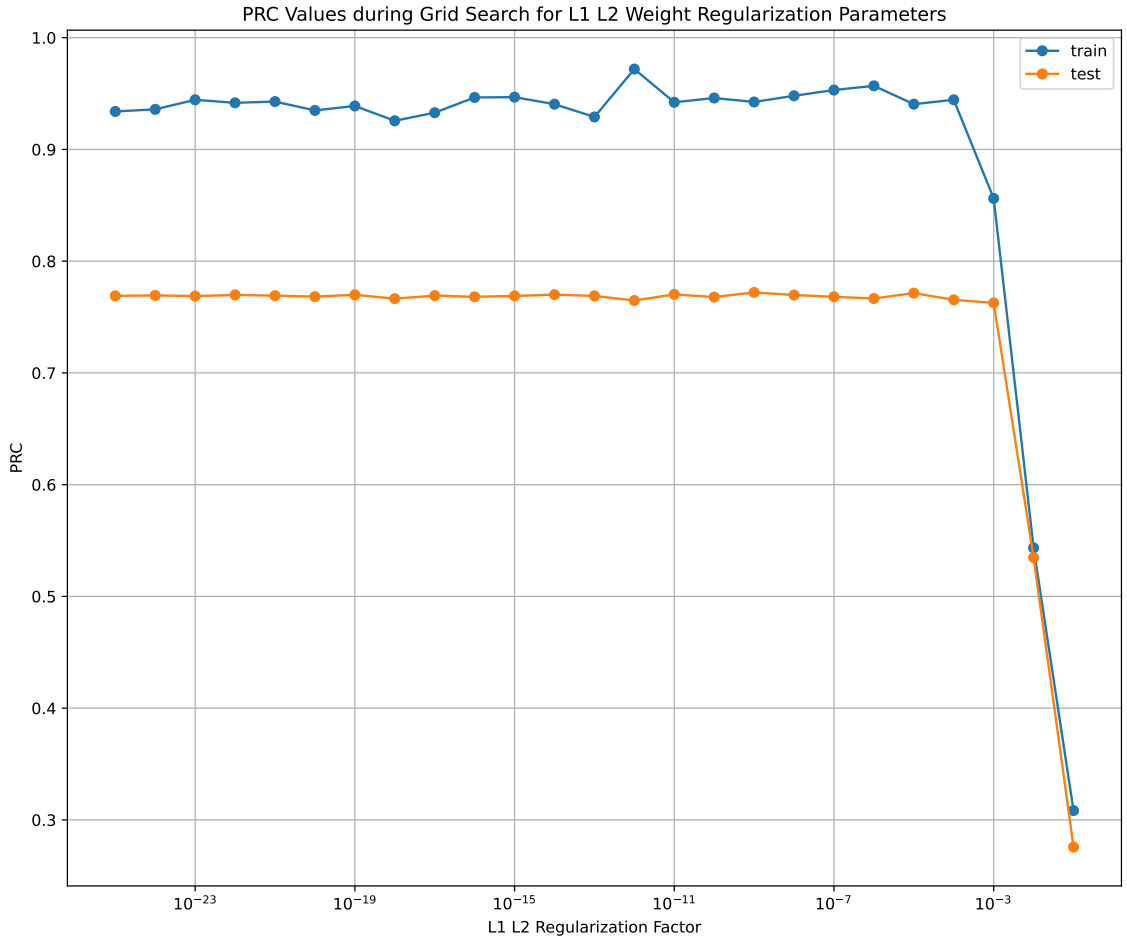


Fig. 3.6 Grid search for L1 L2 weight regularization on the deep FFNN architecture used for the oversampled training set.

AUC-PRC with a maximization objective, and a patience of 50 epochs to ensure that the training is not stopped at a local optimization minima. At the end of training, model weights are restored from the best epoch, which is considered as the best performance achieved during training, before the model began to overfit on the training set. Finally, the prediction threshold for probability output is set to 0.5, i.e., probability outputs below 0.5 are assigned the class 0, and above 0.5 are assigned the class 1. The AUC plots, however, visualise the values and trade-offs of the model behaviour, should the threshold be varied.

3.7 Results and Discussion

Table 3.2 presents the results for the performance of the machine learning methods and base deep neural network model on the multi-label task as of SLO violation prediction. The deep feed-forward neural network performs better than all other machine learning methods on most metrics, and is followed closely by the decision tree method. However, when comparing the averaged AUC-ROC for these models with the performance on individual class labels as shown in Figure 3.3, we notice that almost all the machine learning models have the ROC curves for one or more class labels lying on the diagonal, or close to the diagonal line, which represents a random classifier. This is especially true for the class label SLO_4, i.e. SLO_{io}, which none of the machine learning models learnt well. Hence, the values on precision, recall, and related example and label-based metrics stem from stochastic classification on some class labels, which is undesirable for the use-case here. Further, using such machine learning based methods is suitable only in the cases in which the class labels in the data either have a low level of correlation between them, or are independent altogether. However, it is clear from the comparison with the base deep neural network model that the class labels are correlated, and in a way that is not straightforward to learn given the level of imbalance in the multi-label dataset. Therefore, while binary relevance is the most straightforward in the way it handles the multi-label data, it is limited by its assumptions of complete label independence, which makes it unsuitable for advanced applications. The Classifier Chain method overcomes the limitations of the independent label assumption of the one-vs-rest binary relevance methods, and considers correlations among labels in a random manner. However, the chaining property still has its disadvantages in that it is not a parallel implementation, and the performance is highly dependent on finding the most appropriate order of chaining [106], which random ordering may not always solve. It is also very sensitive to skewed class distributions [107], and thus limited in applications depending on the use-case and the size and characteristics of data.

ML-kNN also assumes label independence, and moreover, takes a lot of time to train. It is therefore unsuited for data of large magnitude. Decision trees are yet another family

of methods that assume label independence in solving multi-label problems. While the biggest advantage of using a decision tree methodology is that it is a white-box method and thus the decisions can be traced back to the logic behind them; on our use-case, the decision tree model holds a depth of 35 with 589 leaves. This conveys a high degree of model complexity that is not as easy to analyze and interpret. Random forest is an ensemble of decision trees, and thus prone to the same issues that decision trees face here.

The deep feed-forward neural network has a clear edge on the task, stemming from the fact that it is capable of modeling the patterns behind each of the class labels in parallel, and can concurrently capture the correlations between them. As shown in Figure 3.8a, the base FFNN model learns the behavior of all classes, including class SLO_4. However, a parallel view of the individual classes for the base FFNN model in Table 3.5 shows that unlike the other class labels, SLO_{io} has no true positive or false positive detections. To address this, we improve the model with class weighting and oversampling strategies, as elaborated earlier in Section 3.5. Table 3.5 presents the performance of these strategies as compared and evaluated on the multi-label classification metrics. A direct inference from the comparison is a quantification of how inflated the performance estimates from the base model were. By adding strategies to address class imbalance, we prevent the model from developing a bias towards the majority class labels, and ensure that the prediction is not an overestimation.

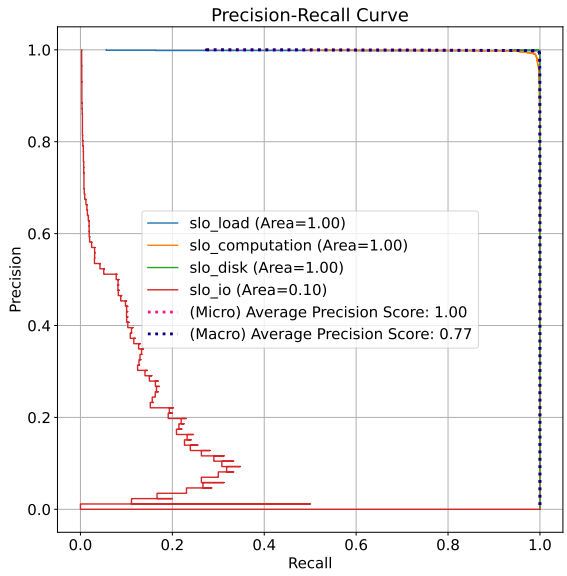
As mentioned earlier in Section 3.5, rather than training to maximize accuracy or any other conventional metric, we train the deep neural network models to maximize the area under the precision recall curve (AUC-PRC). Figure 3.7 presents the Precision-Recall curves for these deep FFNN models. A high AUC-PRC implies high recall and high precision, i.e. low false negative rate and low false positive rate respectively.

Figure 3.8 depicts the ROC curves for the individual class labels, . Any point on the curve here signifies a trade-off between precision and recall, should that be the threshold— if set low, the recall of the positive occurrence (class value 1) will be high, and the precision will be low; and vice-versa if the threshold is set high. Which one to prioritize depends on the use-case— for example in our case the model can be tolerant of false positives at the cost of minimizing the false negatives, since false negatives signify missed SLO violations, which have a higher cost associated with them. Thus, in our use-case, we prioritize recall over precision when measuring performance. Figure 3.9 gives an overview of the degree of overfitting within the AUC-ROC curves in effect in each of the trained neural network models.

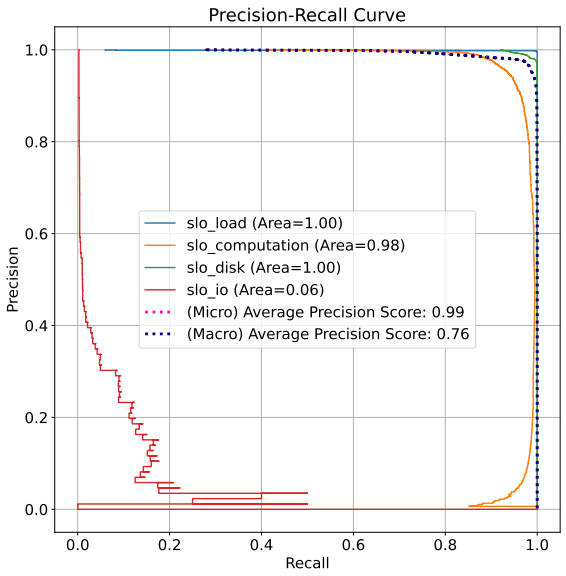
Figures 3.11, 3.12, 3.13, 3.14, and 3.15 show the learning curves depicting loss, AUC-PRC, precision, and recall performance for the corresponding deep FFNN models as they train to converge over the training and validation sets. When comparing the class weighting strategy, we notice the fluctuations when the model assumes large weights as per the

Table 3.5 Performance of the various strategies adopted to address the class label imbalance, evaluated on all presented multi-label classification metrics. Best numeric values corresponding to each metric have been highlighted in bold.

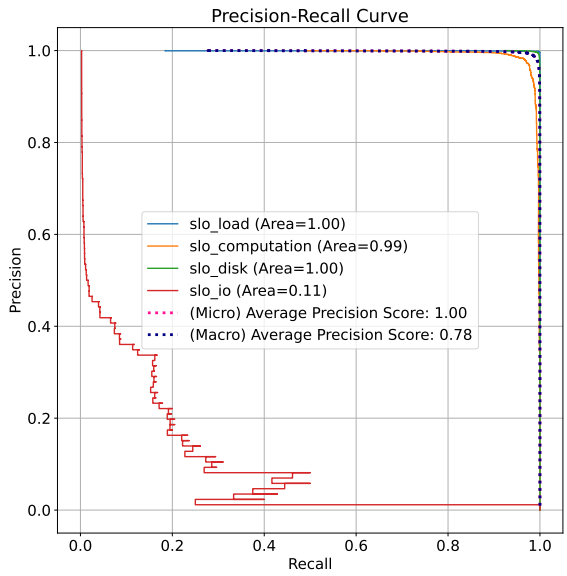
				Addressing Class Imbalance							
		Deep FFNN		**Deep FFNN with Balanced Class Weights**		**Deep FFNN with Clipped Class Weights**		**Deep FFNN with Log Smoothed Class Weights**		**Deep FFNN with Random Oversampling**	
Training Loss		0.009		0.519		0.020		**0.010**		0.046	
AUC-PRC	Micro Average	1.000		0.837		**0.999**		**0.999**		**0.999**	
	Macro Average	0.778		0.741		0.771		**0.776**		0.773	
	Weighted Average	0.998		0.952		0.997		**0.998**		0.997	
AUC-ROC	Micro Average	1.000		0.953		0.999		**1.000**		**1.000**	
	Macro Average	0.969		0.885		0.936		**0.959**		0.955	
	Weighted Average	1.000		0.757		0.997		**0.999**		**0.999**	
True Positives (TP)	SLO 1 (Load)	1973	38040	1974	12131	1976	37678	1973	38024	1972	**38030**
	SLO 2 (Computation)	3499		3529		3517		3491		3499	
	SLO 3 (Disk)	32568		6591		32168		32547		32528	
	SLO 4 (IO)	0		37		17		13		31	
False Positives (FP)	SLO 1 (Load)	1	91	9	2846	18	367	0	**150**	1	355
	SLO 2 (Computation)	37		1818		240		35		93	
	SLO 3 (Disk)	53		514		37		81		59	
	SLO 4 (IO)	0		505		72		34		202	
True Negatives (TN)	SLO 1 (Load)	33429	103322	33421	100567	33412	103046	33430	**103263**	33429	103058
	SLO 2 (Computation)	31822		30041		31619		31824		31766	
	SLO 3 (Disk)	2749		2288		2765		2721		2743	
	SLO 4 (IO)	35322		34817		35250		35288		35120	
False Negatives (FN)	SLO 1 (Load)	5	179	4	26088	2	541	5	195	6	**189**
	SLO 2 (Computation)	50		20		32		58		50	
	SLO 3 (Disk)	38		26015		438		59		78	
	SLO 4 (IO)	86		49		69		73		55	
Precision	Micro Average	0.998		0.810		0.990		**0.995**		0.991	
	Macro Average	0.747		0.663		0.779		**0.813**		0.776	
	Weighted Average	0.996		0.904		0.991		**0.994**		**0.994**	
Recall	Micro Average	0.996		0.317		0.986		**0.996**		0.995	
	Macro Average	0.747		0.656		0.794		0.775		**0.835**	
	Weighted Average	0.996		0.317		0.986		**0.996**		0.995	
F-1 Score	Micro Average	0.997		0.456		0.988		**0.996**		0.993	
	Macro Average	0.747		0.560		0.786		0.787		**0.793**	
	Weighted Average	0.996		0.409		0.988		**0.995**		0.994	
Jaccard Similarity Coefficient	Micro Average	0.994		0.295		0.976		**0.991**		0.986	
	Macro Average	0.744		0.478		0.753		0.764		**0.765**	
	Weighted Average	0.994		0.282		0.978		**0.992**		0.991	
(Subset) Accuracy (or EMR)		0.993		0.197		0.974		**0.990**		0.984	
Hamming Loss		0.001		0.204		0.006		**0.002**		0.003	
Log Loss		0.289		1.680		0.312		**0.294**		0.303	
Subset Zero-One Loss		0.006		0.802		0.025		**0.009**		0.015	
Coverage		1.080		1.186		1.081		**1.080**		**1.080**	
Average Precision (Label Ranking)		0.999		0.946		**0.999**		**0.999**		**0.999**	
Ranking Loss		0.0002		0.035		0.0006		**0.0003**		0.0004	



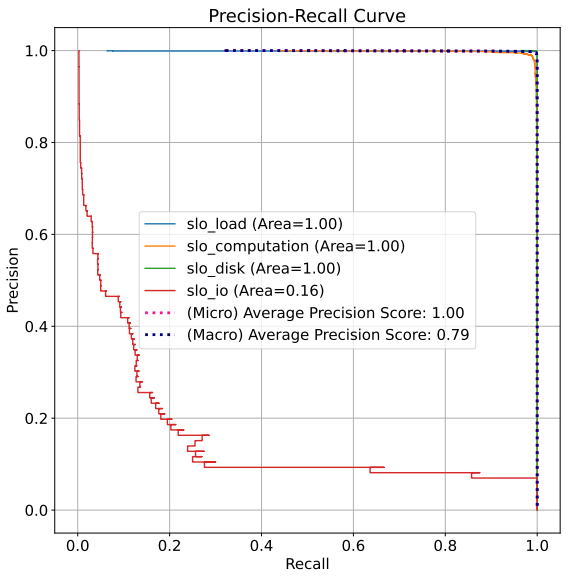
(a) Base Neural Network Classifier— Deep Feed-Forward Neural Network Model



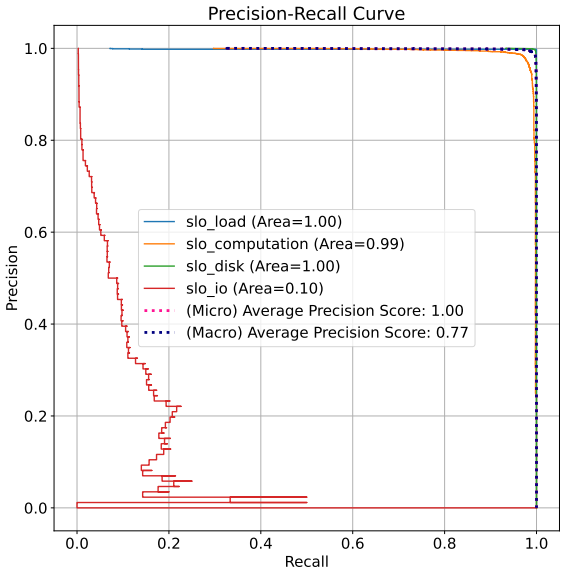
(b) Deep Feed-Forward Neural Network Model— Balanced Class Weights



(c) Deep Feed-Forward Neural Network Model— Clipped Class Weights



(d) Deep Feed-Forward Neural Network Model— Log Smoothed Class Weights

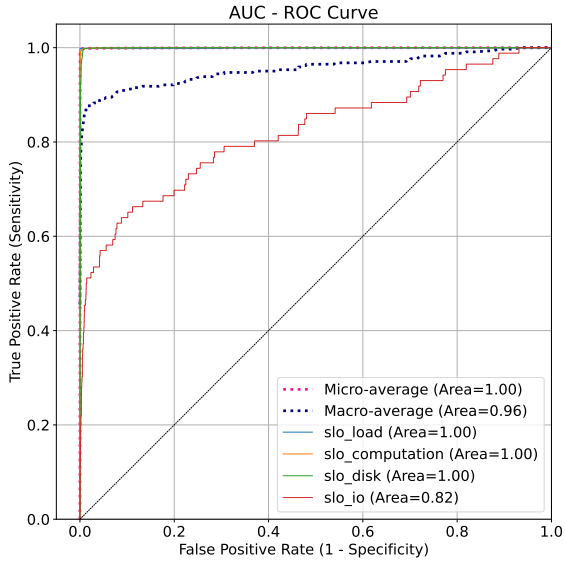


(e) Deep Feed-Forward Neural Network Model—
With Random Oversampling

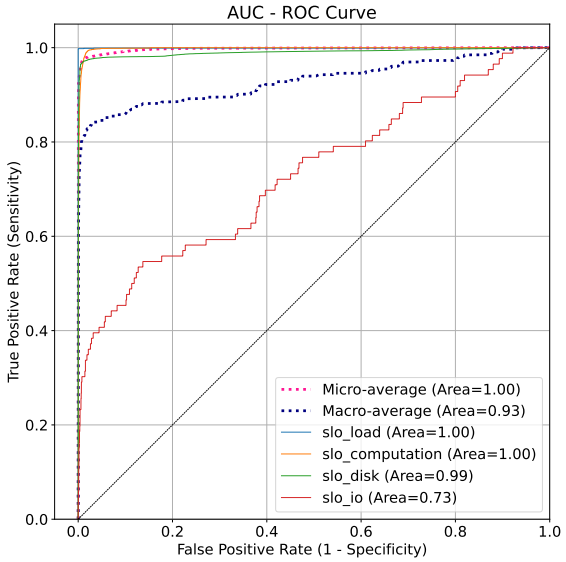
Fig. 3.7 AUC-PRC plots depicting the learning and precision-recall trade-off of the various deep neural network classifiers on individual class labels, as well as the micro and macro averaged performance on the entire set.

balanced class weighting strategy. This is because the model sees an infrequent example associated with a very large weight within some batch of training, which suddenly disrupts the gradient signal at each occurrence. This also transfers to poor performance for all evaluations on the test set. Clipping the class weights to a lower range improves both model training and performance, thus necessitating that formal class weighting strategies that enforce large weights given a high ratio of imbalance are unsuitable in this context. The best performance of the model is achieved when the weights are smoothed by log scaling, and kept to a minimal ratio. The log smoothed model outperforms on almost all categories of metrics with the best convergence and least overfitting, and achieves the highest macroaveraged AUC-PRC (0.776) and AUC-ROC (0.959), with a subset accuracy of 99%, and multi-label accuracy of 99.1%. Figure 3.10 gives an overview of the degree of overfitting within the AUC-PRC curves in effect in each of the trained neural network models.

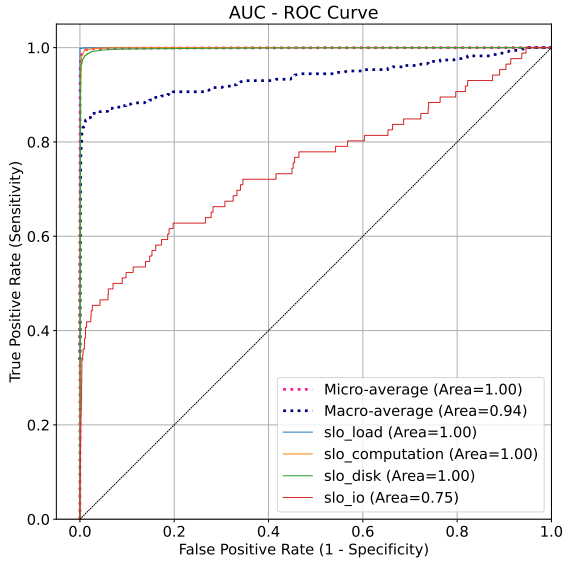
As can be seen in Figure 3.15, the model with random oversampling does overfit given the repetitions in the training set, but the appropriate regularization ensures that the validation curve is smooth, and the model converges well. The model has the highest macroaveraged F_1 score (0.793) and recall (0.835), and fares almost at par with the log weighted model in learning and perform well on all classes. However, while it minimizes the false negatives the most, it has a slightly higher number of false positives. The



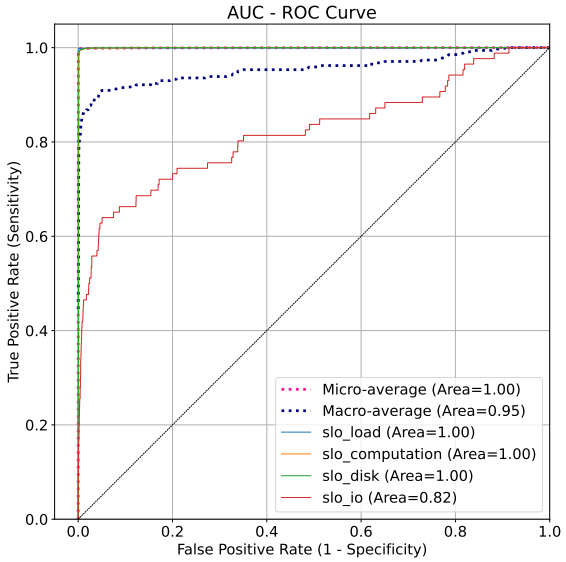
(a) Base Neural Network Classifier— Deep Feed-Forward Neural Network Model



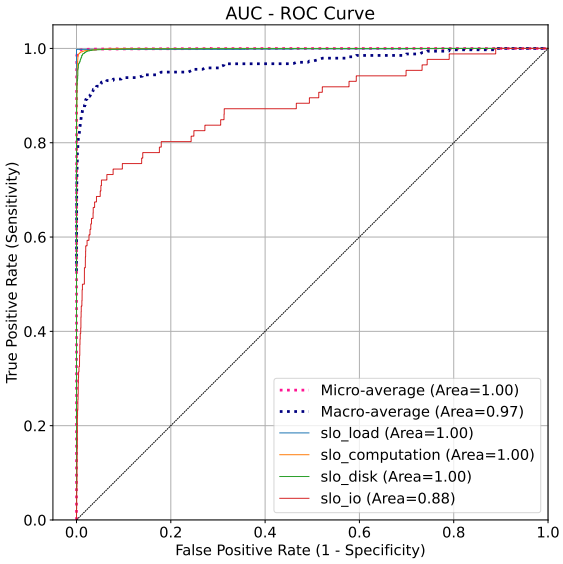
(b) Deep Feed-Forward Neural Network Model— Balanced Class Weights



(c) Deep Feed-Forward Neural Network Model— Clipped Class Weights



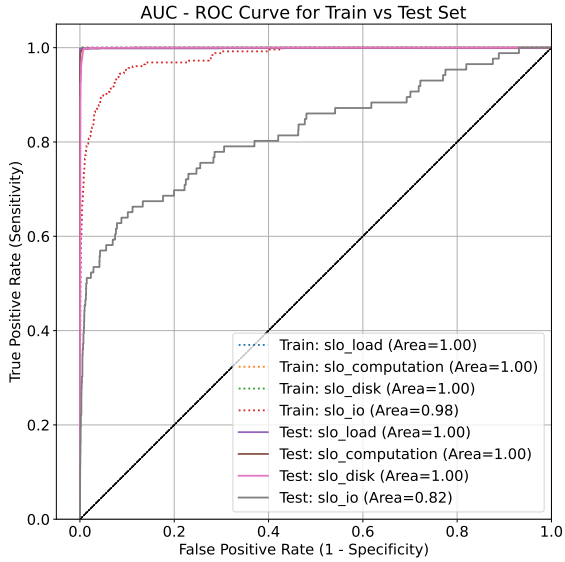
(d) Deep Feed-Forward Neural Network Model— Log Smoothed Class Weights



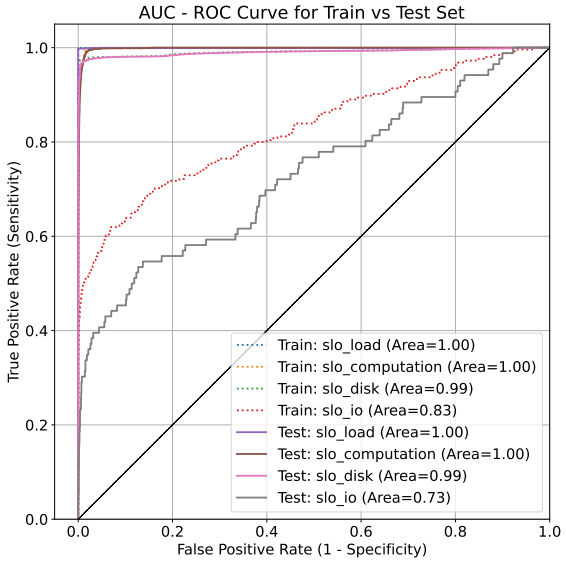
(e) Deep Feed-Forward Neural Network Model—
With Random Oversampling

Fig. 3.8 AUC-ROC plots depicting the learning of the various deep neural network classifiers on individual class labels, as well as the micro and macro averaged performance on the entire set.

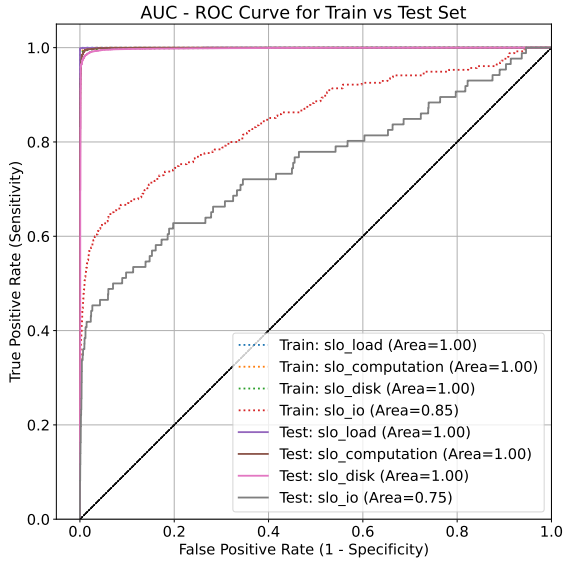
exact classification distributions for the individual class labels are depicted in Table 3.5. Ultimately, the choice between opting for sampling based strategies or class weighting strategies depends on the degree of imbalance, the model being used, the use-case and objectives. In the case of a deep FFNN model, when the class weights enforced are small, both class weighting and oversampling work similarly, assigning the equivalent of small weights to infrequent positive examples in individual batches during training. However, when the class weights formally attain large values, the results suggest that oversampling may be better strategy, since it involves a smoother gradient update in each batch seen during training.



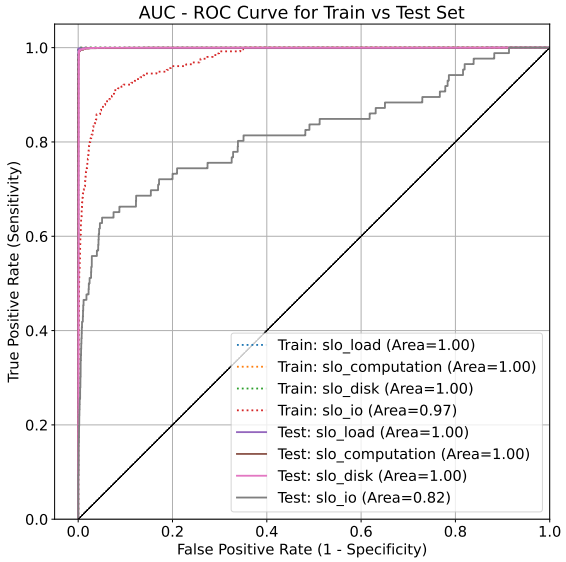
(a) Base Neural Network Classifier— Deep Feed-Forward Neural Network Model



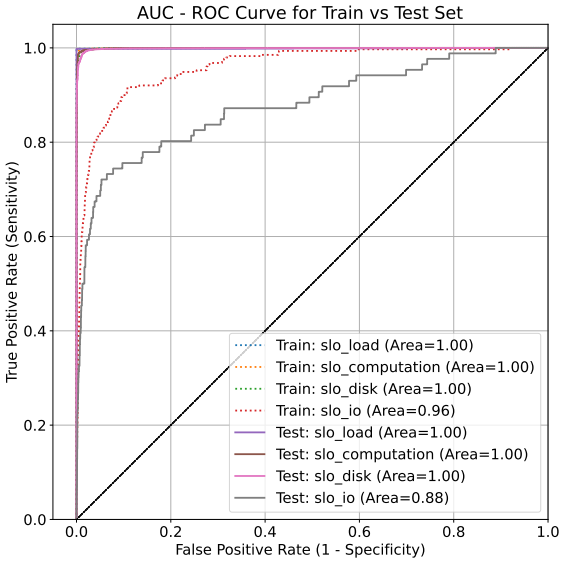
(b) Deep Feed-Forward Neural Network Model— Balanced Class Weights



(c) Deep Feed-Forward Neural Network Model— Clipped Class Weights

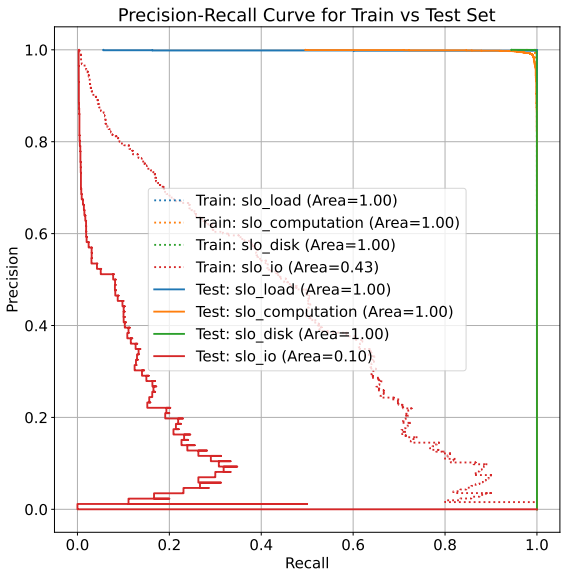


(d) Deep Feed-Forward Neural Network Model— Log Smoothed Class Weights

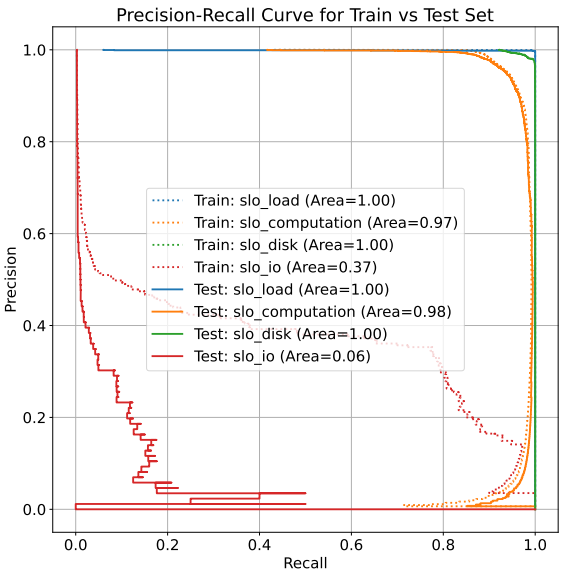


(e) Deep Feed-Forward Neural Network Model—
With Random Oversampling

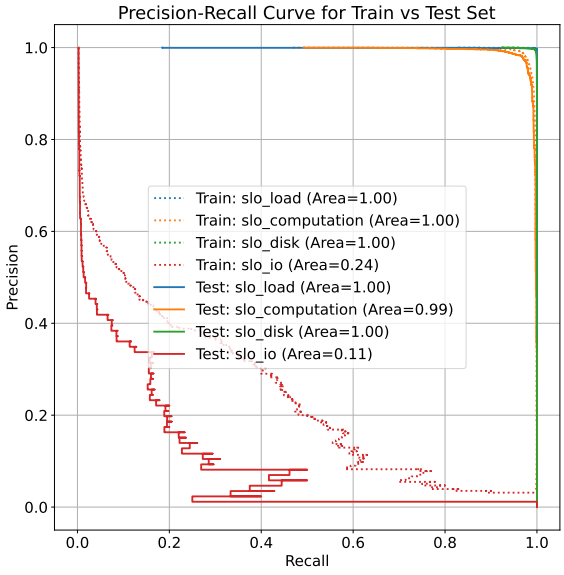
Fig. 3.9 AUC-ROC plots depicting the performance of the various deep neural network classifiers on individual class labels on the Train vs Test set, signifying the degree of overfitting.



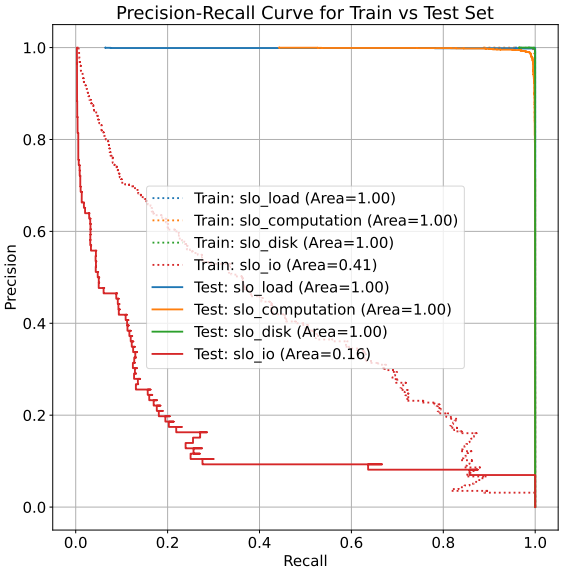
(a) Base Neural Network Classifier— Deep Feed-Forward Neural Network Model



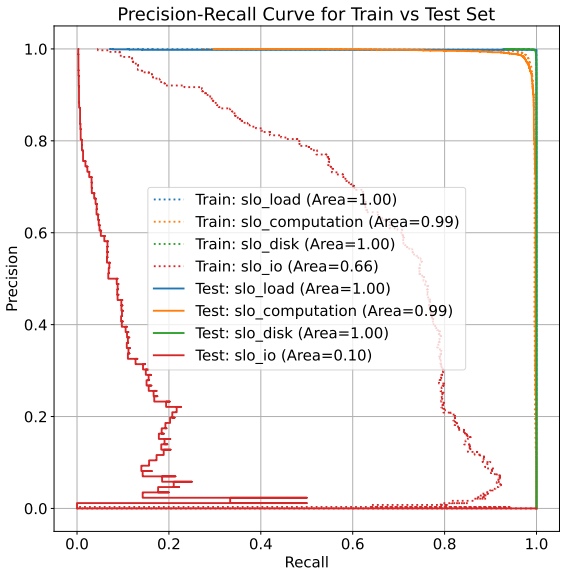
(b) Deep Feed-Forward Neural Network Model— Balanced Class Weights



(c) Deep Feed-Forward Neural Network Model— Clipped Class Weights



(d) Deep Feed-Forward Neural Network Model— Log Smoothed Class Weights



(e) Deep Feed-Forward Neural Network Model—
With Random Oversampling

Fig. 3.10 AUC-PRC plots depicting the performance of the various deep neural network classifiers on individual class labels on the Train vs Test set, signifying the degree of overfitting.

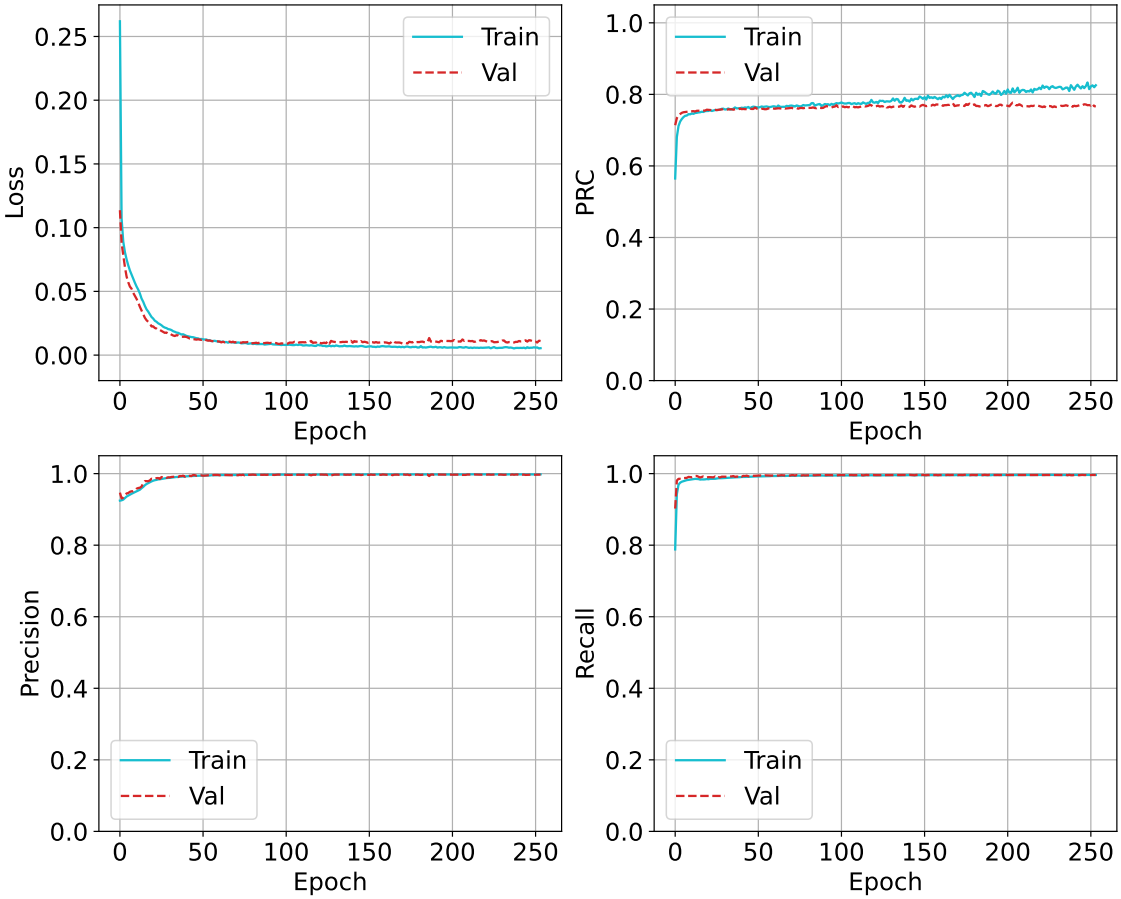


Fig. 3.11 Learning Curves depicting training and validation performance for Base Neural Network Classifier— Deep Feed-Forward Neural Network Model

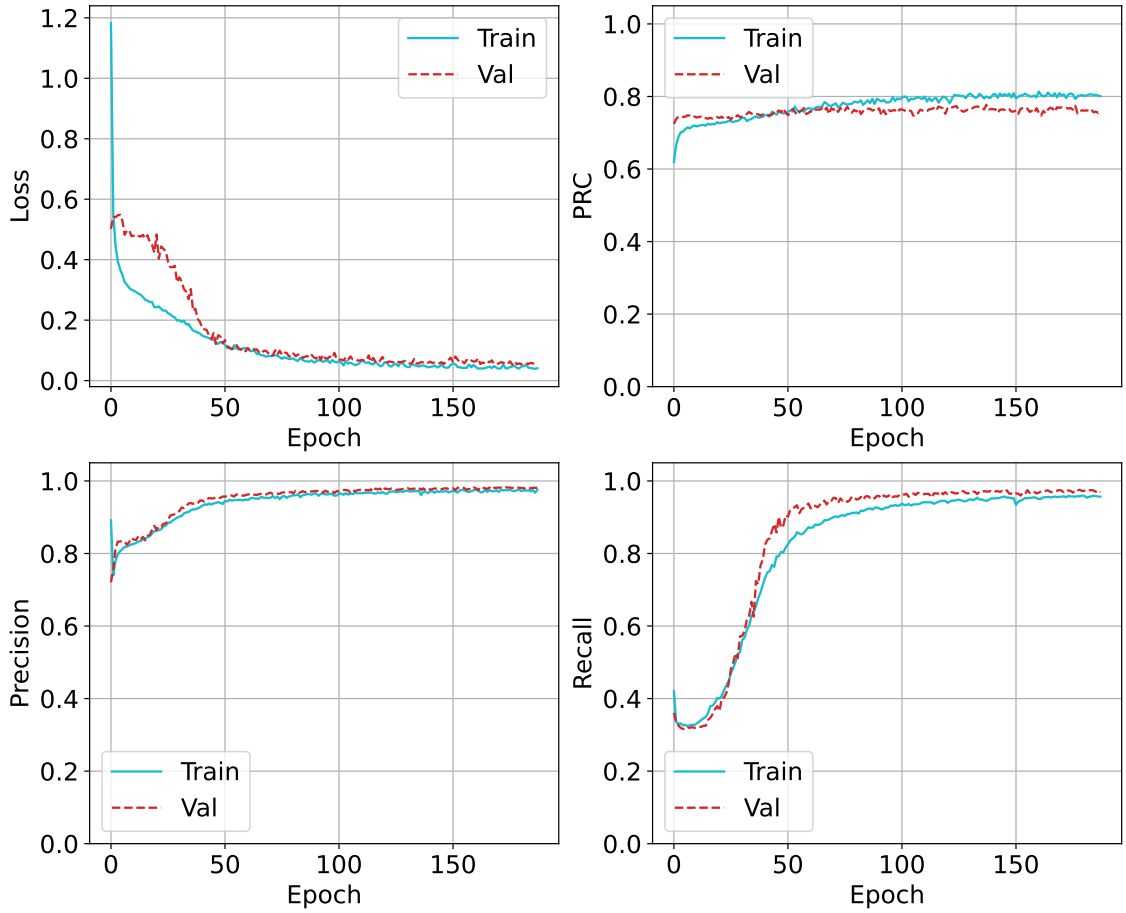


Fig. 3.12 Learning Curves depicting training and validation performance for Deep Feed-Forward Neural Network Model— Balanced Class Weights

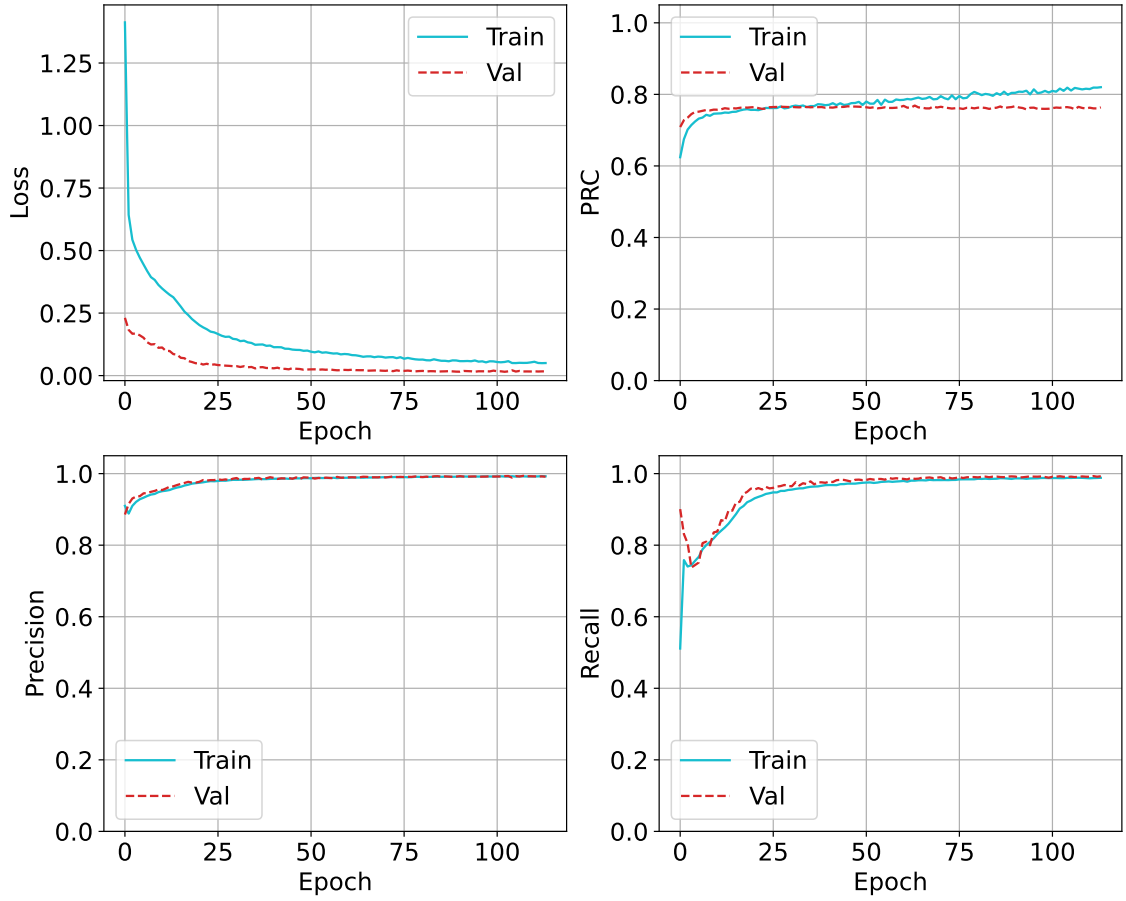


Fig. 3.13 Learning Curves depicting training and validation performance for Deep Feed-Forward Neural Network Model— Clipped Class Weights

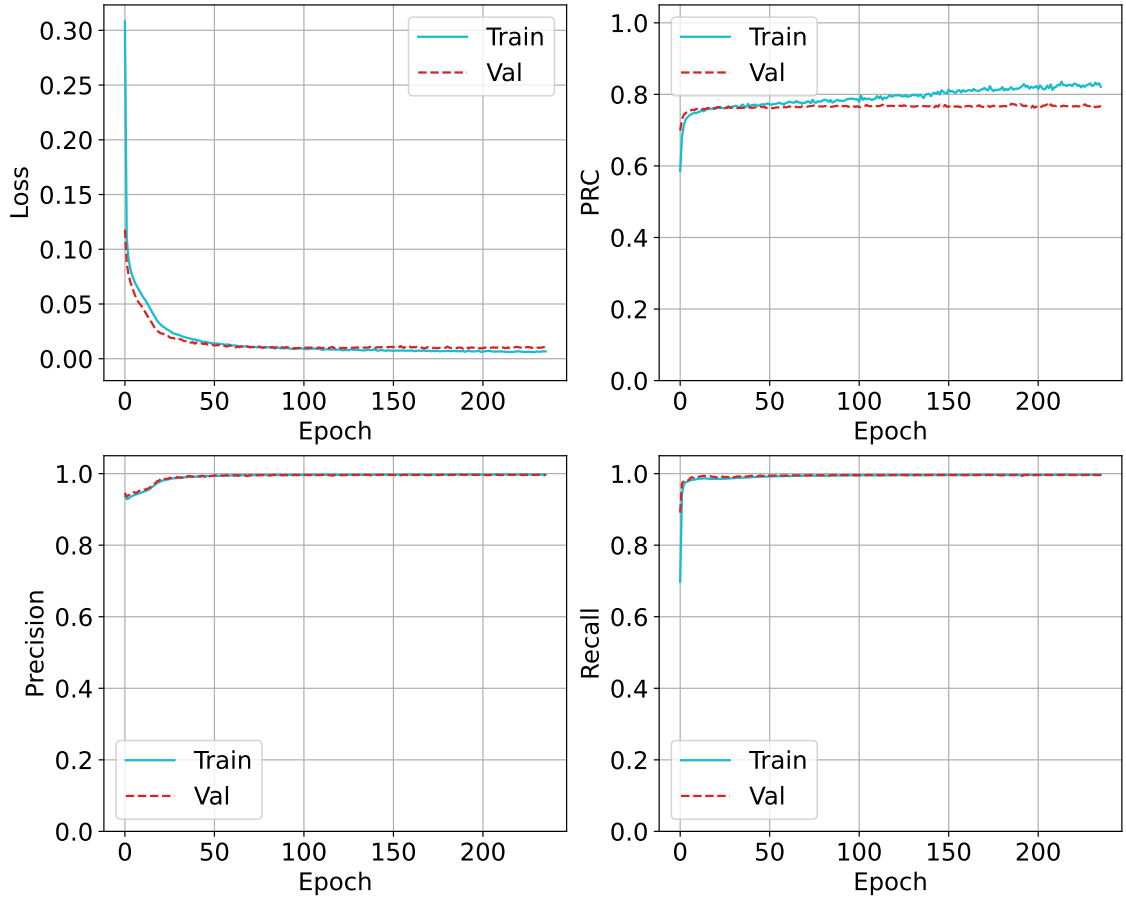


Fig. 3.14 Learning Curves depicting training and validation performance for Deep Feed-Forward Neural Network Model— Log Smoothed Class Weights

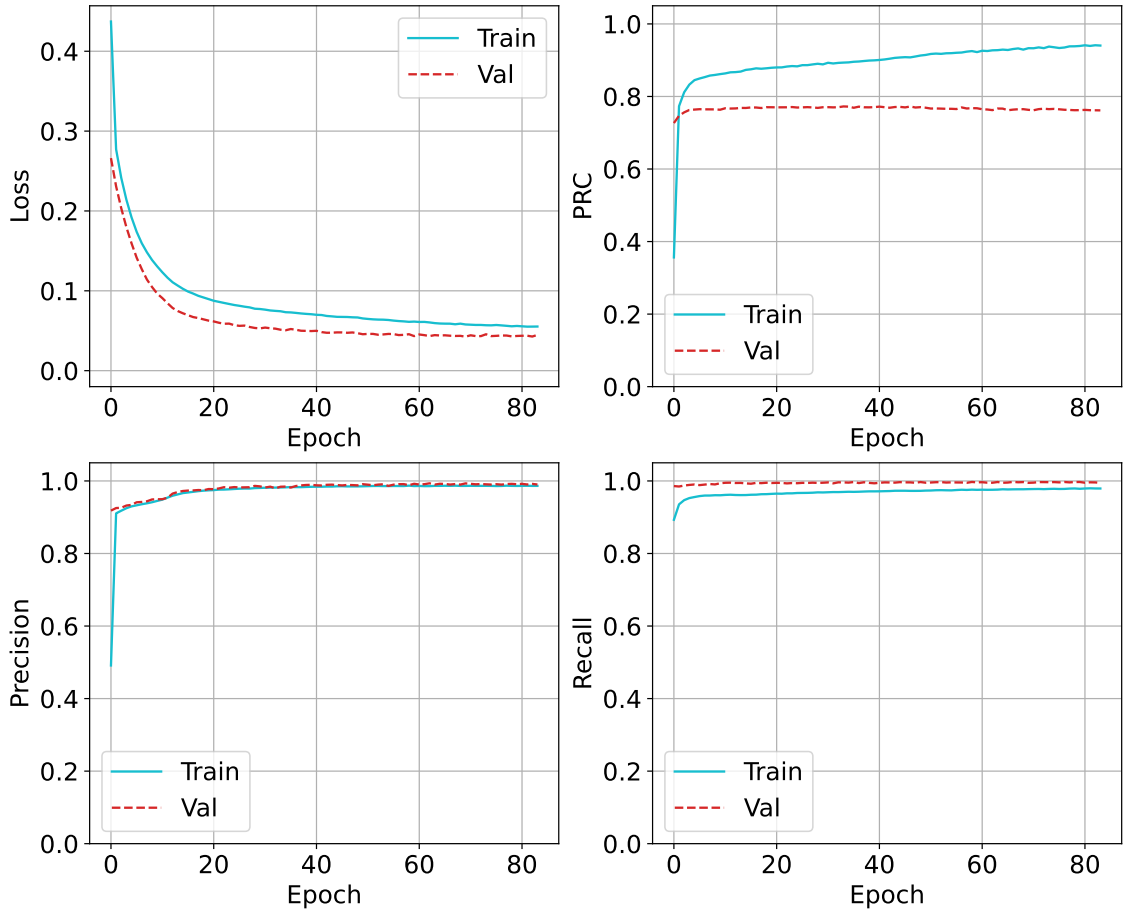


Fig. 3.15 Learning Curves depicting training and validation performance for Deep Feed-Forward Neural Network Model— With Random Oversampling

3.8 Summary and Conclusion

In this Chapter, we address the problem of SLA and SLO violation prediction in an NFV environment with the use of multi-label classification methodology. This enables the incorporation of multi-output models as we move towards more complex decision-making in the management of virtualised communication networks, identifying and predicting multiple categories of SLO breaches as applicable to study and mitigate their impact towards SLA and SLO violations. We work with Clearwater, a latency-sensitive NFV based vIMS application to draft realistic SLO definitions for this vertical, and use these as the basis to model the violations as a multi-output target. We propose the use of a deep feed-forward neural network classifier to adequately capture and learn the correlations between the different categories of SLO violations, and predict them as they (co)-occur given the state of the NFV application at a point in time.

The results suggest the suitability of such a deep learning methodology in achieving the target objective, and in also overcoming the issue of class imbalance in training by adapting class weighting and random oversampling strategies to a multi-label setup. We achieve a subset accuracy of 99%, and multi-label accuracy of 99.1% in the best model approach, working with a dataset where the highest class label imbalance ratio is 395.18, with a mean imbalance ratio of 105.40.

We reason and demonstrate that our proposed methodology can be useful to identify the gaps in SLA policy enforcement, to further fine-tune the scaling policies for an enhanced balance between efficiency and reliability, as well as to identify and address the frequent vulnerabilities and bottlenecks that a latency-sensitive real-time application such as this may face.

Chapter 4

A Residual LSTM based Multi-Label Classification Framework for Proactive SLA Management in a Latency Critical NFV Application Use-Case

4.1 Introduction

As mentioned in the pervious Chapters, efficiency and reliability are competing elements within an SLA, marking a trade-off between system usage and requirements [16]. While network operators and Cloud operators may resort to over-provisioning to match the high requirements for these latency-critical applications in a high availability network slice, such practices are inefficient in the long run [7]. A transition towards complete softwarization of networks and services brings in the requirement to adopt more complex models to guarantee QoS and reliability [17]. This is because of an impending evolution in not just the way networks are composed and managed, but also renewed application architectures, corresponding QoS and SLA management techniques, and optimization and automation to cope with the added complexity [21]. A key aspect to driving such a change is in how the Cloud reacts to such a latency-critical demand, and in being precisely proactive over time [7].

In this Chapter, we present a machine learning based framework for proactive SLA management in the use case of a latency-critical NFV application. The key contributions are summarised as follows:

- We work with data from a real-world deployment of a latency critical NFV application with two months' worth of raw network telemetry data sampled every 30 seconds, and use that as the basis for all our policy formation and learning models.

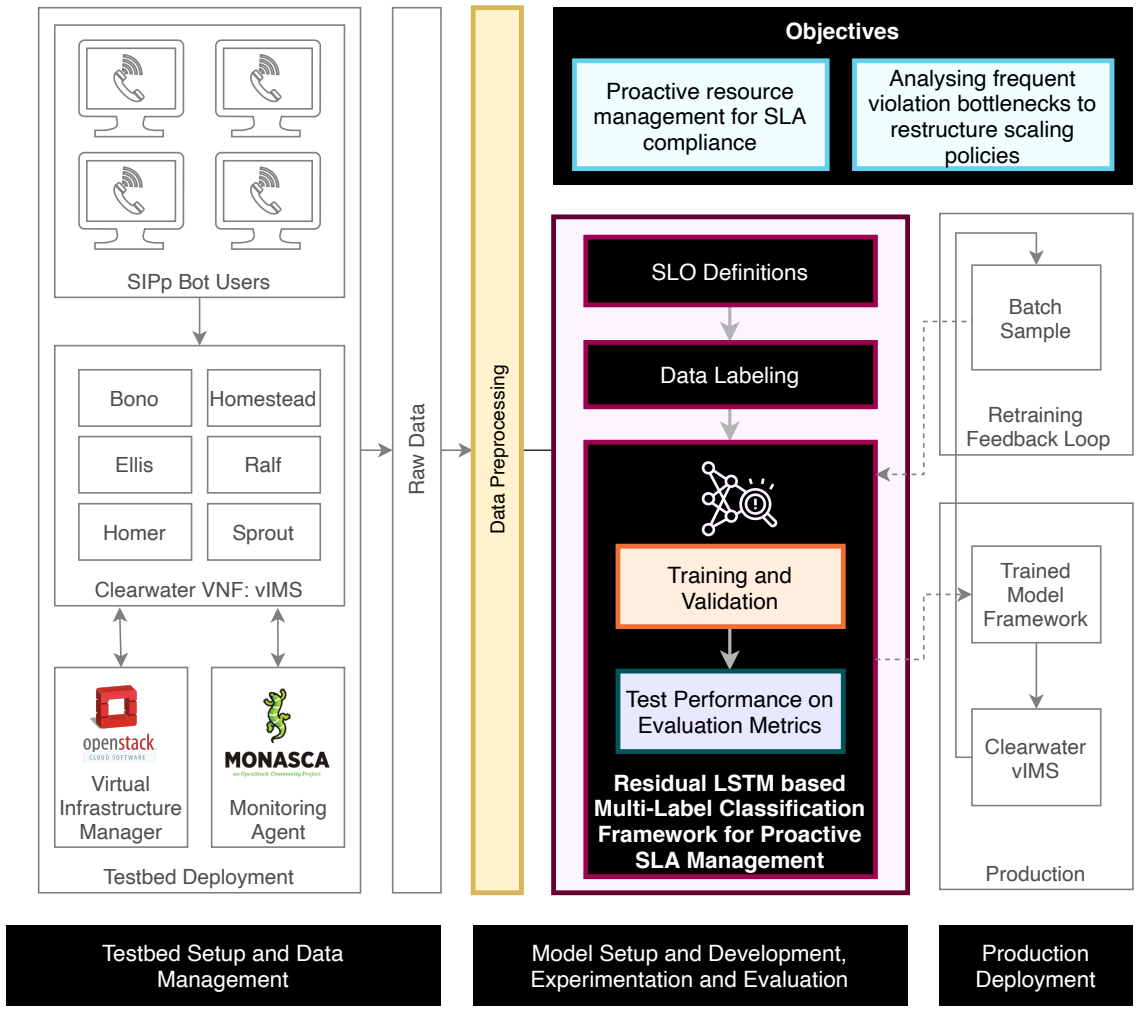


Fig. 4.1 Overview of system architecture, and objectives. The highlighted elements define the scope of our contribution.

An overview of the system and scope is provided in Figure 4.1. The data-set is further elaborated upon in 4.5.1

- We compose a multivariate time-series forecasting model with multiple time-step predictions in a multi-output scenario, i.e. the model forecasts a sequential range of future values for multiple features in one go, thereby enabling us to track a realistic deployment setup. Further, we propose the suitability of a residual connections based Long Short-Term Memory (LSTM) architecture when used for such a task, and compare the performance of multiple forecasting methodologies on such a use-case.
- While drafting the SLA, we decompose it into a set of realistic Service Level Objective (SLO) definitions for such a latency critical use-case in an operational setting. We categorise the SLOs into four broad characteristics that are critical towards the deliverance of required performance, and enhance these for a fine-grained monitoring of a latency-sensitive application that needs high availability and reliability. Further, we associate and model a multi-label classifier to effectively predict each of the multiple SLO violation categories that an application state can concurrently be associated with at an instance, i.e. as a multi-output prediction target. This helps in proactively predicting a more granular state of impact within an SLA violation projection.

To the best of our knowledge, this is the first approach in the area that proposes and applies a Residual LSTM based framework for proactive SLA management, and applies multi-label classification towards such predictive objectives.

The rest of the Chapter has been structured as follows: §4.3 describes the Clearwater NFV application, and defines the SLA and SLOs drafted for the purpose of violation prediction. §4.4 provides an overview of the proposed framework. Thereafter, §4.5 expands on the details of the experimental setup, §4.6 evaluates the results obtained through the various models, and §4.7 presents the summary and conclusion.

The work presented in this Chapter has been disseminated in **[Publication 2 – IEEE CCNC 2022 [9]]**.

4.2 Focus and Scope

This Chapter addresses a component of the third research question (RQ3), i.e.

While tracking the application use-case to proactively avoid SLA violations, do learning methods that track and preserve the inherent topological dependencies perform

holistically better than those that work at the higher-level without active knowledge of this metadata?

This Chapter is focused at comparing the performance of learning methods that fall under the domain of classic deep learning for sequential forecasting, i.e. temporal forecasting at the high level *without* active consideration of spatial metadata and topological dependencies. Chapter 5 will then address the other component of this RQ, i.e. comparison with learning methods that track and preserve the inherent topological dependencies.

4.3 Defining Service Level Agreements

As defined in the last Chapter, while an SLA is a qualitative measure that binds the service provider and facilitator into a formally agreed contract ensuring QoS for the end user, this is realised on a set of low level metrics delivered through SLOs and Service Level Indicators (SLIs). The SLIs can be defined as quantitative measures that build upon raw system metrics, which further feed into the SLOs as a quantitatively definitive target range or threshold towards the deliverance of an SLA.

$$SLI \leq target\ threshold \tag{4.1}$$

$$lowerbound \leq SLI \leq upperbound \tag{4.2}$$

The breach of an SLA implies an explicit consequence, often financial; while the SLOs and SLIs are typically measurable indicators that define the policy of tolerance with measurable service characteristics [10].

4.3.1 Project Clearwater Cloud IMS

To recap from the last Chapter, the IP Multimedia Subsystem (IMS) is a reference architecture first defined by the 3GPP for delivering fixed-line and mobile communications applications built on the Internet Protocol (IP) [78]. Project Clearwater[1] is an open-source implementation of IMS in the Cloud, following IMS architectural principles and supporting all of the key standardized interfaces expected of an IMS core network. The web services-oriented design inherent to Clearwater makes it ideal for instantiation within NFV environments as a virtualized VNF. The new Service-Based Architecture adopted by the 5G standards is very closely related to the inherent Clearwater model, and it has been

[1] www.projectclearwater.org

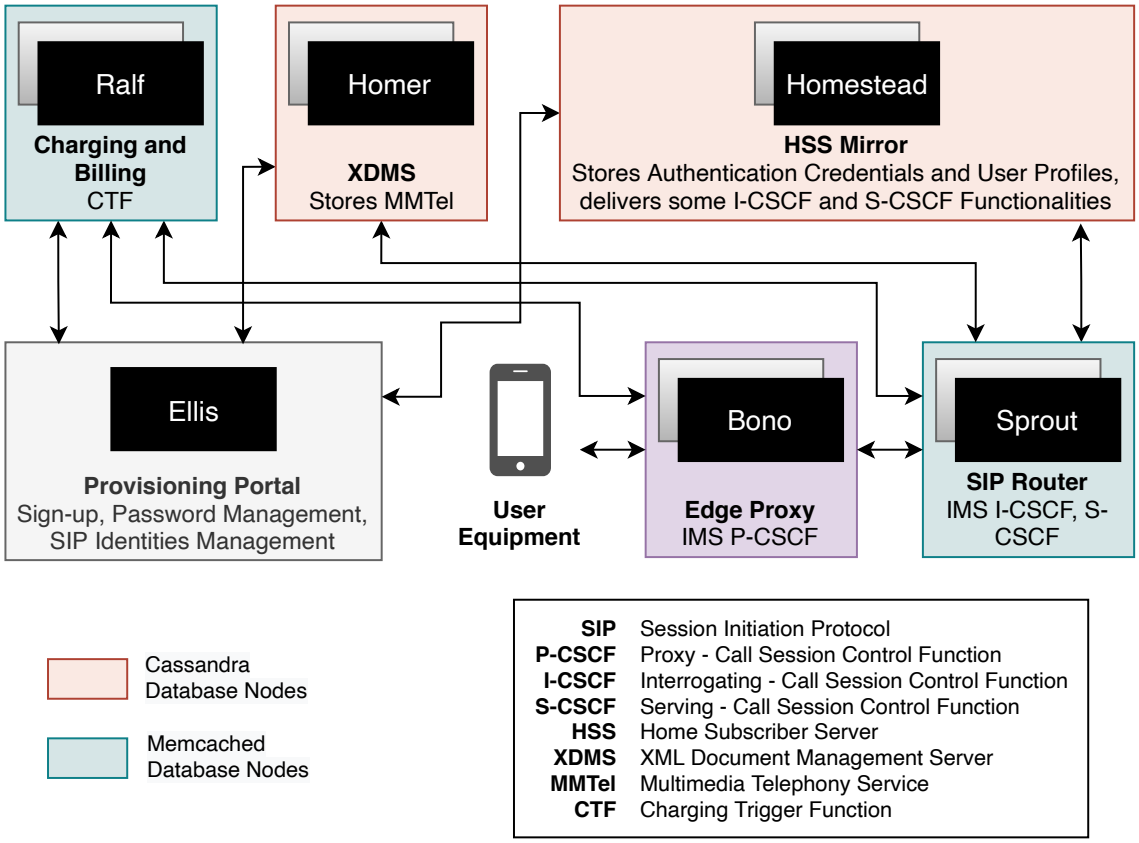


Fig. 4.2 Clearwater vIMS architecture, depicting the various VNFCs and their high-level functionalities.

widely used in research as a standard test-bed setup for NFV related work [6, 7, 16, 17, 42].

In our work, we use Clearwater as the use-case for a Cloud based virtualized NFV application. It consists of 6 main components, namely Bono, Ellis, Homer, Homestead, Ralf, and Sprout. A high level view of these VNFCs and their functionalities replicating a standard IMS architecture is as shown in Figure 4.2.

4.3.2 Defining SLOs for Clearwater

We use raw network telemetry data and system monitoring metrics obtained via a standard realization of the Clearwater test-bed setup to define the SLIs and SLOs governing an informal SLA. We utilise these metrics as the foundations for the SLIs, which when matched with a target threshold or range form SLOs. These metrics were collected on a 30 second sampling frequency through Monasca[2], an open-source Python based monitoring

[2] www.monasca.io

service running on each of the Clearwater VNFCs. Further details regarding the data is elaborated upon in Section 4.5.

As covered in detail in the last Chapter, we define four SLOs for the Clearwater VNF, targeting the load, computation, disk, and input/output (IO) characteristics respectively. This is to highlight the varying reason behind the loss of QoS at any time, so that:

1. The scaling decisions can be dynamically adapted with a high degree of detail, considering the projected forecasts. This helps towards the objective of proactive resource management for SLA compliance.

2. The scaling policies can be customised at a more granular level towards better efficiency, upon analysing long term trends of frequent SLO bottleneck categories as specific to the application.

Authors in [42] recognize the lack of realistic SLOs in consideration in research, and recommend that an SLO be composed of a combination of atleast two metrics. To set a fair ground for our analysis, we define the SLOs with this definition in mind, and use a combination of over two SLIs while drafting each SLO rule. The thresholds were largely defined based on the application's usage characteristics, reaction to stress tests, and use-case requirements.

1. SLO_1*: Load*: This SLO is a measure of the computational work ongoing, and captures the running processes— either using the CPU, or in a wait state.

2. SLO_2*: Computation*: This SLO is defined as a combination of certain CPU and RAM characteristics combined with the idle time profile, which overall characterizes an overload or malfunction.

3. SLO_3*: Disk*: This SLO captures prolonged periods of inefficient IO wait times when the CPU is otherwise idle, which indicates potential bottlenecks in the read/write operations accrued by the hard disk.

4. SLO_4*: IO*: This SLO captures the latency when interacting with IO devices, when there is a sudden and prolonged surge in incoming network traffic as compared to the moving average.

A detailed composition of the SLO definitions can be found in Chapter 3. Formally, let $\mathscr{L}$ denote the set of SLOs thus defined:

$$\mathscr{L} = [SLO_1, SLO_2, SLO_3, SLO_4] \tag{4.3}$$

This equivalently denotes:

$$\mathscr{L} = [SLO_{load}, SLO_{computation}, SLO_{disk}, SLO_{io}] \tag{4.4}$$

The metrics captured by Monasca are at the granularity of the individual VNFCs as shown in Figure 4.2, and an SLO violation at any of the individual VNFCs triggers an SLO violation state for the Clearwater application service. Therefore, we ultimately define the SLOs at the application level, i.e. for the entire VNF as an application service. Thus, each data instance is associated with 4 SLOs as defined by $\mathscr{L}$ above, where $SLO_j, j \in |\mathscr{L}|$ assumes one of two states:

$$SLO_j = \begin{cases} 1, & \text{if } Violated \text{ (at any VNFC)} \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

4.4 Proactive SLA Management Framework

Given that the aim is to be able to proactively predict the future SLA violations given the time-series of tracked system and application metrics at a set sampling frequency, we decompose the problem as that of continuous feature forecasting, followed by a classification methodology that predicts the associated SLO violations in the forecasts. The workflow and pseudo-code of the methodology adopted is as depicted in Algorithm 4.1.

4.4.1 Clearwater Feature Forecasting

Artificial neural networks (ANNs) are powerful non-linear function approximators that are flexible to be adapted to both regression and classification tasks, and have demonstrated tremendous potential within the machine learning space. Each neuron within a layer represents a mathematical function comprising of inputs, weights, bias, and threshold; and uses an activation function to transform the outputs to a non-linear space to learn and perform more complex tasks [100]. Thus, subject to the right choice of architecture and parameters for the task at hand, ANNs can be trained to address a wide variety of complex tasks, including that of time-series forecasting.

Recurrent neural networks (RNN) are a class of neural networks that are powerful for modeling sequential data such as time-series, and are especially crafted towards such use-cases [7]. An RNN layer maintains an internal state that encodes information about the time-steps it has seen so far. LSTM is a special enhancement on RNN, and overcomes the potential gradient vanishing and gradient exploding problems that RNNs tend to

Algorithm 4.1: Residual LSTM based Forecasting and Multi-Label Classification

Input : Data: $\mathscr{D} \in \mathbb{R}^d$

• **PRE-PROCESSING** ($\mathscr{D}$)

procedure DATA SPLITS
  Split the data in train, test, and validation sets
procedure DATA TRANSFORMATION
  Data standardization and normalization
procedure DATA WINDOWING AND BATCHING
  Split data into windows of features and associated labels)
  $<$ `input width, all input features` $>$
  $<$ `label width, predicted features` $>$
  Prepare tensor slices of windows as model inputs
  $<$ `batch, time, features` $>$

• **THE MODEL** (output from DATA WINDOWING)

repeat
  FORECASTING– RESIDUAL LSTM
  Take train and validation data windows
    if `Stacked LSTM model(Return Sequence)` **then**
      Model architecture based sequentially stacked LSTM layers
    else
      Model architecture based LSTM layer
    end if
    for `each time step t` **do**
      delta = $\text{MODEL}_t(\text{MODEL}_{t-1})$
    end for
    return (MODEL_{t-1} + delta)
  Model architecture based Dense layer
  Model architecture based Reshaping layer
  Output forecast values
  MULTI-LABEL CLASSIFICATION
  Dense layer
  Reshaping layer
  Output multi-label classification prediction values
until CONVERGENCE;

succumb to while training using back-propagation [108]. An LSTM layer consists of a set of recurrently connected memory blocks, known as LSTM units. Each LSTM unit is composed of a memory cell and three multiplicative units – the input, output and forget gates. These control the interaction of the cells with the network by regulating the flow of information in and out of the cell with continuous analogues of write, read and reset operations [100].

LSTM models are adept at capturing short-term and long-term dependencies in temporal sequential data, and have been shown to outperform linear models on a wide range of use-case scenarios [34]. Moreover, neural network based approaches like LSTM are well equipped to effectively model problems with multiple input variables, making them a good fit for multivariate time-series forecasting.

However, LSTMs are computationally expensive [32]. In use-cases such as ours with a high sampling frequency and rapid forecasting windows, the output is expected to be a small change as compared to the previous time-step. ResNets (Residual Networks) in deep learning refer to architectures where each layer adds to the model's accumulating result [55]. Adapting that structure into LSTM layer(s) [109], we can take advantage of the fact that the change at the next time-step is expected to be small. Thus, instead of predicting the next value of each feature at each time-step, a better approach to the model structure would be initialize the LSTM layer with the model's values from the previous time-step, and then to predict the subsequent change in these values over the next time-step. We reason that the LSTM layers in our use-case scenario can benefit from such a Residual LSTM model structure, and lead to better performance as opposed to traditional LSTM based models. Algorithm 1 presents the procedural workflow adopted towards achieving such a model. We evaluate this methodology in the following sections for both wide and stacked LSTM model structures.

4.4.2 SLA Violation Classification

Multi-label classification is defined as a classification task where each data sample instance can be assigned n labels from a set of $|\mathscr{L}|$ possible label classes as defined in 4.3 and 4.4, where $n \in [0, \mathscr{L}]$, and $|\mathscr{L}| > 1$. Each of the class labels in $\mathscr{L}$ is binary, i.e. either 0 or 1, where 0 denotes a negative occurrence and 1 denotes the positive occurrence.

Semantically, a multi-label target can be thought of as a set of labels for each sample. Multi-label classification differs from multi-class classification in that the latter applies mutually exclusive labels to a data sample, which is not the case for multi-label problems.

Formally, let $\mathscr{D}$ be a multi-label dataset where $\mathscr{X} = \mathbb{R}^d$ is a d-dimensional input instance space of numerical features, and $\mathscr{L} = \{\lambda_1, \lambda_2, \cdots, \lambda_q\}$ a finite output label space of $|\mathscr{L}| = q$ discrete class labels (with values 0 or 1), and $q > 1$.

The task of multi-label learning is to learn a function $f : \mathscr{X} \longrightarrow 2^{\mathscr{L}}$ from the multi-label training set $\mathscr{S}$ with u examples, $\mathscr{S} = \{(\mathbf{x}_i, Y_i) \mid 1 \leq i \leq u\}$. To compare, multi-class classification can be seen as a special case of multi-label classification where $f : \mathscr{X} \longrightarrow \mathscr{L}$, while in binary classification $f : \mathscr{X} \longrightarrow \{0,1\}$.

For each multi-label example $(\mathbf{x}_i, Y_i)$, $\mathbf{x}_i \in \mathscr{X}$ is a d-dimensional feature vector $(x_{i1}, x_{i2}, \cdots, x_{id})^\top$, and $Y_i \subseteq \mathscr{L}$ is the set of labels associated with $\mathbf{x}_i$. Label associations can also be represented as a q dimensional binary vector $\mathbf{y}_i = (y_{i1}, y_{i2}, \cdots, y_{iq})^\top = \{0,1\}^q$, where each element is 1 if the label is relevant, and 0 otherwise. By contrast, in single-label (binary or multi-class) learning, $|Y| = 1$.

Specific to the task at hand, we appropriately design the model such that the output layer consists of $|\mathscr{L}|$ neurons, each representing a label λ_j in $\mathscr{L}$, where $\mathscr{L} = \{\lambda_j \mid j \in [1,q]\}$. We use sigmoid as the activation function in the output layer, so the j^{th} neuron in that layer outputs the probabilities in the range $[0,1]$ of the data instance belonging to λ_j. This is interpretable as a binary classification by setting a cutoff probability threshold value (set to 0.5) for each class label.

4.5 Experimental Setup

The experiments were all set up using Python (version 3.8.5) and its associated data-science libraries. We use Tensorflow [102] version 2.4.1 with Keras [103] to program all the neural network based implementations.

4.5.1 Dataset

We use a publicly hosted dataset[3] obtained via a standard Clearwater test-bed, a visualization for which is presented in Figure 4.1. While real-world deployments require frequent retraining and readjustment of weights, the nuances of production deployment are beyond the scope of this work. The dataset comprises of raw system resource monitoring telemetric data files that track 26 metrics for each of the 6 monitored VNFs that compose the Clearwater ecosystem, and includes bursts of abnormal behaviour through its integrated stress testing tools to simulate VNF congestions and QoS degradations. The data is sampled every 30 seconds, and spans an overall period of 2 months. This corresponds to 156 features overall, indexed at timestamps, and 177,098 rows of raw data. Outcomes of the exploratory data analysis as performed on this data-set have been presented in Appendix A.

[3] www.bit.ly/3gPY8c5

4.5.2 System Configuration

The experiments were performed in a Docker[4] based containerized environment running atop a bare-metal Linux server with 64 GB RAM, Intel® Xeon® CPU E5-2660 v2 @ 2.20GHz (40 physical processors), 2 NVIDIA® Tesla K20m GPUs, and 500 GB local storage. The Docker image runs an Ubuntu 20.04 LTS operating system, and CUDA version 11.3 for the GPUs.

4.5.3 Learning and Adaptation

We adopt a sliding window methodology for time-series forecasting, which is suitable for rapid forecasting. We tested the models on different window sizes, and considering the short-term dynamics of the use-case, the input window size was optimally set to 4, and the models forecast the possibilities of SLO violations over each of the next 4 time-steps. Thus, since the sampling frequency is 30 second intervals, we use the data for all the features over the last 2 minutes to forecast the specific SLOs that may be violated at each step over the next 2 minutes.

We split the available data into training and test sets in the ratio of 80 : 20, and the training set is further split into training and validation sets in the ratio of 80 : 20. Hence, overall, the data consisting of 177,098 rows is split in the ratio 64 : 20 : 16, corresponding to train, test, and validation splits respectively. Further, given the data has a high degree of very large outliers due to the incorporated stress tests, and the scales vastly vary for each of the features depending on the category of raw metric, we use a non-linear transformation method to transform the very skewed nature of this dataset and map it to a uniform distribution in $[0, 1]$. This pre-processing is done via a Quantile Transformer, and this makes it suitable for learning by neural network methodologies.

For model training, we use Binary Cross-entropy (BCE) as the loss function to be minimized— a probabilistic loss function that computes the cross-entropy loss between true and predicted labels, and is appropriate for use in a binary classification based setup. Further, we use Nadam as the optimizer for its computational efficiency and adaptive learning, with its default learning rate of 10^{-3}. ReLu (Rectified Linear Unit) is used as the activation function for each of the dense hidden layers due to its computational simplicity and high optimization performance in a multi-layer perceptron (MLP) based setup [100]. For the LSTM layers, we use the default tensorflow initializations for weights, activation, and bias. As mentioned earlier, we use sigmoid as the activation function for the output layer to concurrently output the individual probability of each label's association with the input data instance, thus supporting multi-label classification outputs.

[4] www.docker.com

We used a grid search based methodology to arrive at the best configuration for the number of neurons/ LSTM units in each hidden layer as applicable, and to adjust all hyperparameters. To control overfitting as the model gains complexity, we apply the dropout regularization factor of 0.2 between all hidden layers. On the LSTM layers, we also applied a recurrent dropout of 0.4. To further control the degree of overfitting during training, we perform a grid search for the optimal choice of weight regularization hyperparameters for all the hidden layers, and based on the results, apply both L1 and L2 (ElasticNet) weight regularization on each of the hidden layers. The regularization factor was set to 10^{-6} for the LSTM models, and 10^{-5} for the dense feed-forward neural network models.

The batch size for each window was set to 64. While we set the maximum training epochs to 100, we also deploy an early stopping criteria that tracks the macroaveraged AUC-PRC (interpolated average precision) with a maximization objective, and a patience of 10 epochs to ensure that the training is not stopped at a local optimization minima. At the end of training, model weights are restored from the best epoch, which is considered as the best performance achieved during training, before the model began to overfit on the training set.

4.6 Results and Discussion

Figure 4.3 presents an evaluation of Residual LSTM models against standard methods, when configured with the same wide model architecture (2048 neurons/LSTM units), and implemented purely as a forecasting component. The LSTM architecture when supplemented with residual connections has a clear advantage with the nature of the use-case here, both when the output horizon is forecast at the last time-step after going through all inputs, or sequentially. The latter among them, however, has a higher Mean Absolute Error (MAE) due to the fact that the model architecture was wide, and sequential outputs of LSTM layers work best when layers need to be stacked in a deeper model.

Table 4.1 Performance metrics for the best performing model architecture within each category

Model Name	No. of Neurons/Units in the Key Hidden Layers in the Model Architecture	Classification Metrics						Regression Metrics	
		Accuracy	Precision	Recall	AUC-ROC	AUC-PRC	BCE Loss	MAE	RMSE
Linear	130	0.6488	0	0	0.5	0.3512	5.4176	0.3512	0.5926
Dense	2048	0.8500	0.9025	0.6424	0.8074	0.7955	2.0124	**0.1599**	0.3836
LSTM	2048	0.8500	0.9025	0.6424	0.7893	0.7776	1.9082	0.1625	0.3853
Residual LSTM	128	**0.8510**	**0.9082**	**0.6424**	**0.8082**	**0.7982**	1.6747	0.1647	**0.3810**
Residual LSTM (Stacked Sequence)	1024, 512, 256	0.8500	0.9025	0.6424	0.8069	0.7929	**1.3284**	0.1629	0.3839

Figure 4.4 shows the performance benchmarking of the varied model architectures and configurations as evaluated during grid search. Generally, models that leverage residual

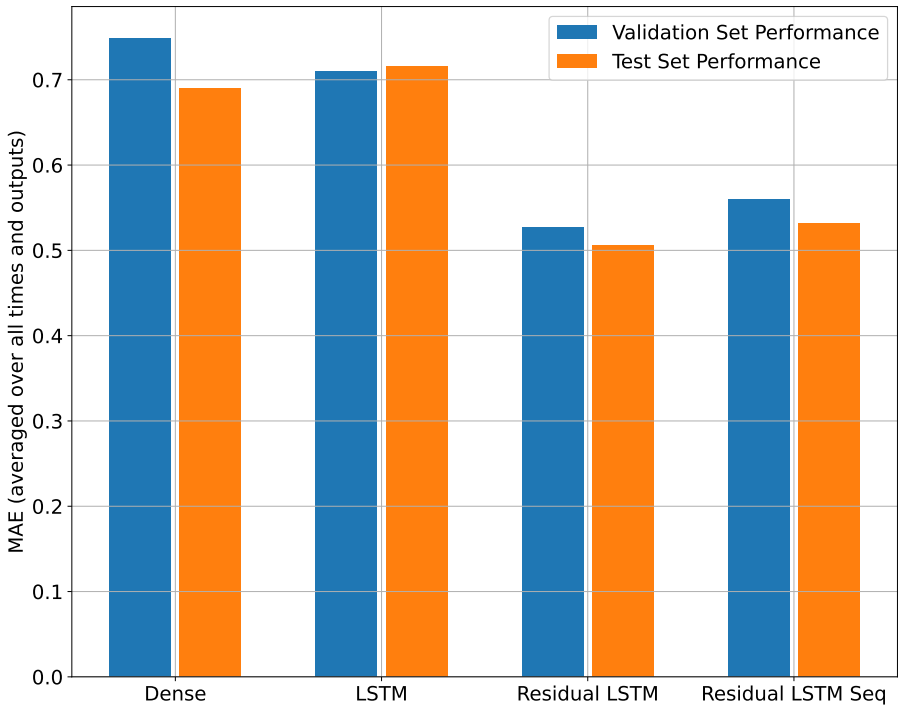
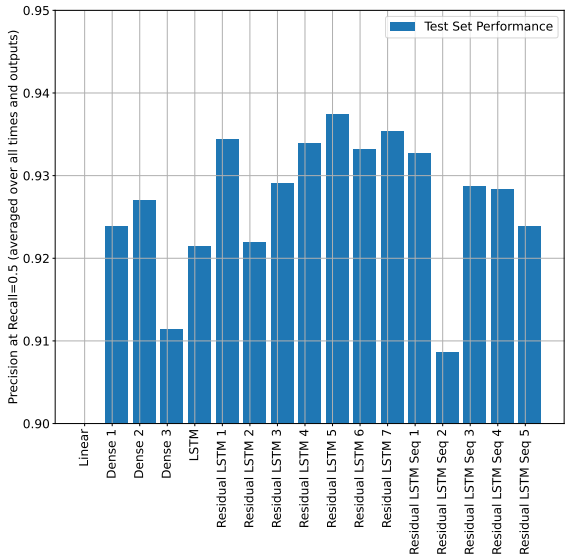


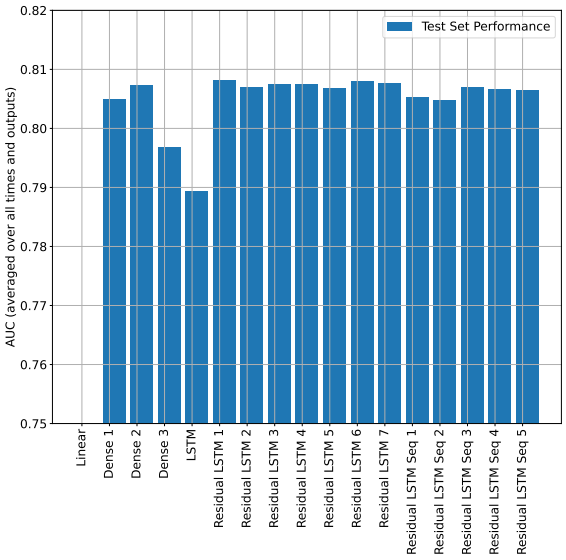
Fig. 4.3 Performance (MAE) of Residual LSTM models evaluated on their forecasting component against standard methodologies, when configured with the same wide model architecture.

connections perform better in almost all model configurations when tested against standard methods. Table 4.1 presents the classification and regression performance metrics for the model architecture that performed best among those tested with each category of models on the test set data windows. The linear model makes linear projections based on the input window, and was used as a baseline to compare all models. With an area under the receiver operating characteristic (AUC-ROC) value of 0.5, it has the skill level of a random classifier. The results demonstrate the suitability of Residual LSTM based architectures against all other, on almost all metric categories in both evaluation groups. The Residual LSTM approach achieves an improvement of 31.1% over the baseline on the forecast classification accuracy, 127.28% on the interpolated average precision, and 61.64% on the AUC-ROC. It also showed a 0.63% increase in precision over the standard LSTM methodology, a 2.65% improvement on AUC-PRC, and 2.39% improvement on the AUC-ROC.

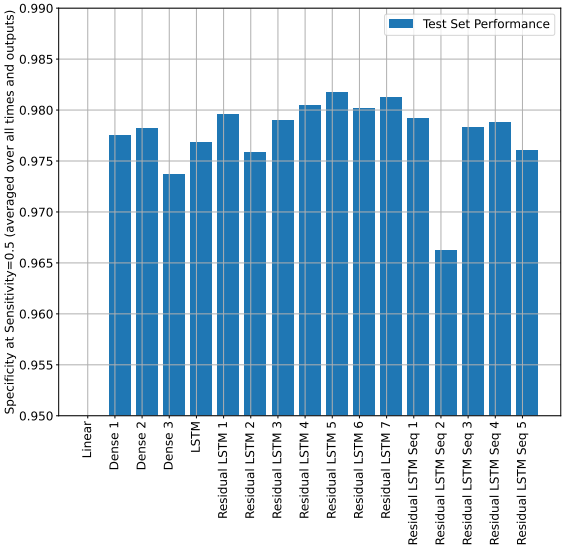
Further, given the use-case and the nature of the data, wide model architectures for LSTM demonstrate an edge over deep architectures. While adding more LSTM units in a layer tends to increase the chances of overfitting, we tackled those with appropriate regularization as applicable, as mentioned in the preceding section.



(a) Precision at Recall=0.5



(b) AUC-ROC



(c) Specificity at Sensitivity=0.5

Fig. 4.4 Performance benchmarking of the varied model architectures and configurations, evaluated within each category.

4.7 Summary and Conclusion

In this Chapter, we compose a multivariate time-series forecasting model that forecasts the evolution of system monitoring features for the Clearwater VNF over the next 4 time steps, followed by a multi-label classification model that predicts the individual categories of SLO violations at each step over a 2 minute future horizon. We demonstrate the suitability of a Residual LSTM model over other MLP and LSTM based methodologies in such a scenario that involves fine-grained rapid forecasting, and reason that the high level of granularity in predicting SLOs as multi-label outputs would help ensure a balance in precise provisioning while maintaining reliability in latency-critical NFV applications.

Chapter 5

A Graph Neural Networks based Framework for Topology-Aware Proactive SLA Management in a Latency Critical NFV Application Use-case

5.1 Introduction

As Cloud infrastructure transitions to serve the next-generation requirements of upcoming softwarized application verticals [3], this is met with evolutionary challenges of incremental reliability guarantees [4]. As highlighted in the previous Chapters, efficiency and reliability are competing elements within an SLA, marking a trade-off between system usage and requirements [16]. While network operators and Cloud operators may resort to over-provisioning to match the high requirements for these latency-critical applications in a high availability network slice, such practices are inefficient in the long run [7]. A transition towards complete softwarization of networks and services brings in the requirement to adopt more complex models to guarantee Quality of Service (QoS) and reliability [17]. This is because of an impending evolution in not just the way networks are composed and managed, but also renewed application architectures, corresponding QoS and SLA management techniques, and optimization and automation to cope with the added complexity [21]. A key aspect to driving such a change is in how the Cloud reacts to such a latency-critical demand, and in being precisely proactive over time [7].

In this Chapter, we present a graph-based deep learning framework for dynamic and proactive SLA management in the use case of a latency-critical NFV application. The key contributions are summarised as follows:

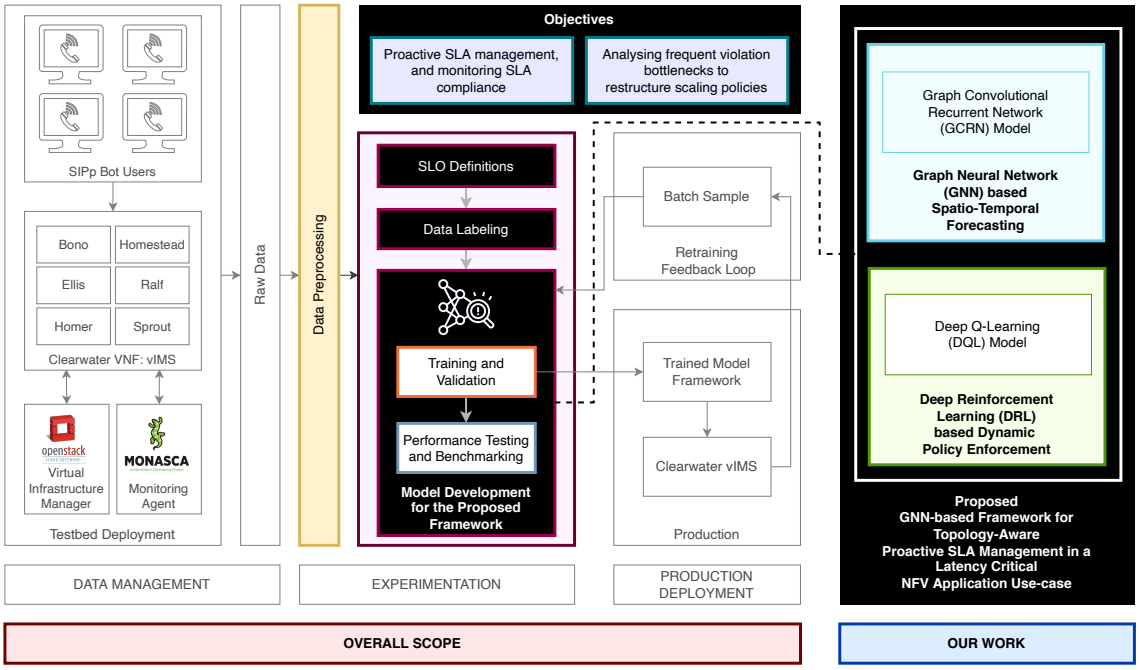


Fig. 5.1 Overview of system architecture, objectives, and highlighted scope.

- As with the previous Chapters, we work with data from a real-world deployment of a latency critical NFV application with two months' worth of raw network telemetry data sampled every 30 seconds, and use that as the basis for all our policy formation and learning models. An overview of the system and scope is provided in Figure 5.1. The data-set is further elaborated upon in 5.5.1

- We compose a multivariate time-series forecasting model with multiple time-step predictions in a multi-output scenario, i.e. the model forecasts a sequential range of future values for multiple features in one go, thereby enabling us to track a realistic deployment setup. Further, we propose the suitability of a topology-aware Graph Neural Network (GNN) based model as opposed to traditional Recurrent Neural Network (RNN) methodologies when applied to such a task. Specifically, our implementation of a Gated Recurrent Unit (GRU) based Graph Convolutional Recurrent Network (GCRN) model demonstrates a 74.62% improvement in performance over the established state-of-art model [9] on the use-case.

- We leverage realistic Service Level Objective (SLO) definitions defined in our previous work [8] (and Chapter 3) to compose a Q-learning based Deep Reinforcement Learning (DRL) model to achieve dynamic SLA-aware policy enforcement for such a latency critical use-case in an operational setting.

To the best of our knowledge, this is the first approach in the area that proposes a GNN and DRL based framework for proactive SLA management, and widely outperforms the

previously established residual connections based Long Short-Term Memory (Residual-LSTM) model benchmark [9] in the use-case's predictive objectives.

The rest of the Chapter has been structured as follows: §5.3 describes the Clearwater NFV application, and defines the SLA and SLOs leveraged in the framework. §5.4 provides an overview of the proposed framework. Thereafter, §5.5 expands on the details of the experimental setup, §5.6 evaluates the results obtained through the various models, and §5.7 presents the conclusion and future work.

The work presented in this Chapter has been disseminated in **[Publication 3 – IEEE Access]**, currently under review. An early proof of concept for combining reinforcement learning with traditional graph neural networks (GNN) was disseminated in **[Publication 4 – IEEE NFV-SDN 2019 [7]]**. While its contents are not directly a part of this Chapter, the publication has been placed in Appendix B for reference.

5.2 Focus and Scope

This Chapter addresses the third research question (RQ3), i.e.

While tracking the application use-case to proactively avoid SLA violations, do learning methods that track and preserve the inherent topological dependencies perform holistically better than those that work at the higher-level without active knowledge of this metadata?

and the fourth research question (RQ4), i.e.

How can reinforcement learning enable a dynamic SLA-aware policy enforcement control loop, working towards an adaptable system with automated model tuning and dynamic readjustment?

5.3 Defining Service Level Agreements

To recap from the earlier Chapters, while an SLA is a qualitative measure that binds the service provider and facilitator into a formally agreed contract ensuring QoS for the end user, this is realised on a set of low level metrics delivered through SLOs and Service Level Indicators (SLIs). The SLIs can be defined as quantitative measures that build upon raw system metrics, which further feed into the SLOs as a quantitatively definitive target range or threshold towards the deliverance of an SLA.

$$SLI \leq target\ threshold \tag{5.1}$$

$$lowerbound \leq SLI \leq upperbound \tag{5.2}$$

The breach of an SLA implies an explicit consequence, often financial; while the SLOs and SLIs are typically measurable indicators that define the policy of tolerance with measurable service characteristics [10].

5.3.1 Project Clearwater Cloud IMS

As mentioned in previous Chapters, the IP Multimedia Subsystem (IMS) is a reference architecture first defined by the 3GPP for delivering fixed-line and mobile communications applications built on the Internet Protocol (IP) [78]. Project Clearwater[1] is an open-source implementation of IMS in the Cloud, following IMS architectural principles and supporting all of the key standardized interfaces expected of an IMS core network. The web services-oriented design inherent to Clearwater makes it ideal for instantiation within NFV environments as a virtualized VNF. The new service-based architecture adopted by the 5G standards is very closely related to the inherent Clearwater model, and it has been widely used in research as a standard test-bed setup for NFV related work [6, 7, 16, 17, 42].

In our work, we use Clearwater as the use-case for a Cloud based virtualized NFV application. It consists of 6 main components, namely Bono, Ellis, Homer, Homestead, Ralf, and Sprout. A high level view of these VNFCs and their functionalities replicating a standard IMS architecture is as shown in Figure 5.2.

5.3.2 Defining SLOs for Clearwater

Same as with the previous Chapters, we use raw network telemetry data and system monitoring metrics obtained via a standard realization of the Clearwater test-bed setup to define the SLIs and SLOs governing an informal SLA. We utilise these metrics as the foundations for the SLIs, which when matched with a target threshold or range form SLOs. These metrics were collected on a 30 second sampling frequency through Monasca[2], an open-source Python based monitoring service running on each of the Clearwater VNFCs. The list of these collected metrics is presented in Table 5.1, and further details regarding the data is elaborated upon in Section 5.5.

In our previous work [8] as also presented in Chapter 3, we fulfill the gap of a lack of realistic SLOs when used in research, and define Clearwater SLOs by using a combination

[1] www.projectclearwater.org
[2] www.monasca.io

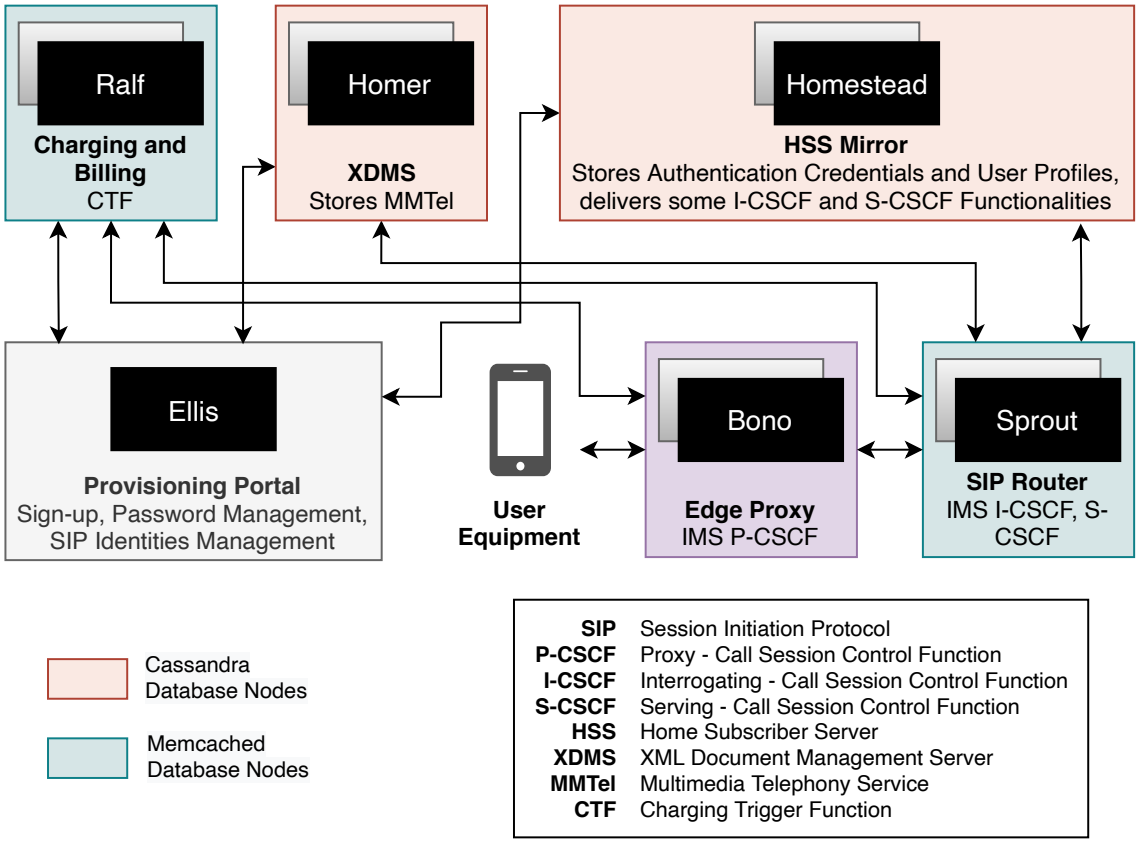


Fig. 5.2 Clearwater vIMS architecture, depicting the various VNFCs and their high-level functionalities.

Table 5.1 Raw metrics collected through Monasca during Clearwater vIMS application monitoring. This data is available for each of the VNFCs, and is sampled every 30 seconds.

		Metric Name	Semantics
CPU		cpu.idle_perc	Percentage of time the CPU is idle when no IO requests are in progress
		cpu.system_perc	Percentage of time the CPU is used at the system level
		cpu.wait_perc	Percentage of time the CPU is idle AND there is at least one IO request in progress
Disk	**Disk**	disk.inode_used_perc	The percentage of inodes that are used on a device
		disk.space_used_perc	The percentage of disk space that is being used on a device
	IO Read	io.read_kbytes_sec	Kbytes/sec read by an IO device
		io.read_req_sec	Number of read requests/sec to an IO device
		io.read_time_sec	Amount of read time in seconds to an IO device
	IO Write	io.write_kbytes_sec	Kbytes/sec written by an IO device
		io.write_req_sec	Number of write requests/sec to an IO device
		io.write_time_sec	Amount of write time in seconds to an IO device
Load		load.avg_1_min	The normalized (by number of logical cores) average system load over a 1 minute period
		load.avg_15_min	The normalized (by number of logical cores) average system load over a 15 minute period
		load.avg_5_min	The normalized (by number of logical cores) average system load over a 5 minute period
Memory		mem.free_mb	Mbytes of free memory
		mem.usable_mb	Total Mbytes of usable memory
		mem.usable_perc	Percentage of total memory that is usable
Network	**In**	net.in_bytes_sec	Number of network bytes received per second
		net.in_packets_sec	Number of network packets received per second
	Out	net.out_bytes_sec	Number of network bytes sent per second
		net.out_packets_sec	Number of network packets sent per second

of over two SLIs while drafting each SLO rule [42]. To highlight the varying reason behind the loss of QoS at any time, we define four SLOs for the Clearwater VNF, targeting the load,

computation, disk, and input/output (IO) characteristics respectively. As detailed in Chapter 3, the thresholds were largely defined based on the application's usage characteristics, reaction to stress tests, and use-case requirements.

1. SLO_1*: Load*: This SLO is a measure of the computational work ongoing, and captures the running processes— either using the CPU, or in a wait state.
2. SLO_2*: Computation*: This SLO is defined as a combination of certain CPU and RAM characteristics combined with the idle time profile, which overall characterizes an overload or malfunction.
3. SLO_3*: Disk*: This SLO captures prolonged periods of inefficient IO wait times when the CPU is otherwise idle, which indicates potential bottlenecks in the read/write operations accrued by the hard disk.
4. SLO_4*: IO*: This SLO captures the latency when interacting with IO devices, when there is a sudden and prolonged surge in incoming network traffic as compared to the moving average.

Formally, let $\mathscr{L}$ denote the set of SLOs thus defined:

$$\mathscr{L} = [SLO_1, SLO_2, SLO_3, SLO_4] \tag{5.3}$$

This equivalently denotes:

$$\mathscr{L} = [SLO_{load}, SLO_{computation}, SLO_{disk}, SLO_{io}] \tag{5.4}$$

The metrics captured by Monasca are at the granularity of the individual VNFCs as shown in Figure 5.2, and an SLO violation at any of the individual VNFCs triggers an SLO violation state for the Clearwater application service. Therefore, we ultimately define the SLOs at the application level, i.e. for the entire VNF as an application service. Thus, each data instance is associated with 4 SLOs as defined by $\mathscr{L}$ above, where $SLO_j, j \in |\mathscr{L}|$ assumes one of two states:

$$SLO_j = \begin{cases} 1, & \text{if } \textit{Violated} \text{ (at any VNFC)} \\ 0, & \text{otherwise} \end{cases} \tag{5.5}$$

5.4 Proactive SLA Management Framework

Given that the aim is to be able to proactively mitigate the potential SLA violations given the time-series of tracked system and application metrics at a set sampling frequency,

Algorithm 5.1: A GNN-based Framework for Topology-Aware Proactive SLA Management in a Latency Critical NFV Application Use-case

• **PRE-PROCESSING**
Input : Data: $\mathscr{D} \in \mathbb{R}^d$
procedure DATA SPLITS
| Split the data in train, test, and validation sets
procedure DATA TRANSFORMATION
| Data standardization and normalization
procedure DATA WINDOWING AND BATCHING
| Split data into sliding windows of features and targets

• **GCRN FORECASTING MODEL**
Input : Train and validation data windows from DATA WINDOWING
Define Clearwater VNFC graph *nodes*
Initialize Clearwater graph *edge indices* (connections)
repeat
| • X = Input data features sliding tensor slice
| • Y = Expected data features sliding tensor slice
| • $\hat{Y}$ = Forecasted data features sliding tensor slice
| • GCRN Model Layers (in, out, K):
| GConvGRU layer (Clearwater *edge indices*) per Equation 5.7
| Linear layer
| • Return: MSE loss between $(\hat{Y}, Y)$
until *Convergence*;
Output : Forecasted features $\hat{Y}$

• **DEEP REINFORCEMENT LEARNING MODEL**
Input : $x_t \in \mathscr{D}$
Initialize Q network function with random weights θ
Initialize target $\hat{Q}$ function with random weights $\hat{\theta}$
Initialize experience replay memory $\mathscr{E}$ to set capacity

repeat
| **TRAINING DEEP Q-NETWORK**
| • Observe state s_t (i.e. x_t) of Clearwater forecasting environment at time t
| • Policy π: ε-greedy
| **if** Exploration (with probability ε) **then**
| Randomly select action a_t
| **else**
| Select action $a_t = \text{argmax}_a Q(s_t, a; \theta)$
| **end if**
| • Record reward r_t for action a_t
| • Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathscr{E}$
| • Sample random minibatch of transitions (s_i, a_i, r_i, s_{i+1}) from $\mathscr{E}$
| • Perform a gradient descent step on $(r_i - Q(s_i, a_i; \theta))^2$ with respect to θ
| • Reset $\hat{Q} = Q$ periodically
until *Training Episode Counter Complete*;
Output : Trained reinforcement learning model

we decompose the problem as that of continuous feature forecasting, followed by a reinforcement learning methodology that oversees the scaling policy to avoid potential SLA violations based on the forecasts. The workflow and pseudo-code of the methodology adopted is as depicted in Algorithm 5.1.

5.4.1 Graph Neural Network (GNN) based Clearwater Feature Forecasting

Recurrent neural networks (RNN) are a class of neural networks that are powerful for modeling sequential data such as time-series, and are especially crafted towards such use-cases [7]. An RNN layer maintains an internal state that encodes information about the time-steps it has seen so far. However, traditional forecasting by way of evolved RNN models like GRU and LSTM consider only temporal information of features for sequence modeling.

When considering use-cases with additional spatial dependencies between features, relying solely on temporal variation likely imposes a cap on performance no matter how complex the RNN model in use. When input data can inherently be represented in a graph-based format of nodes and edges, the temporal flow of information between nodes also involves spatial dependencies. GNNs are a family of neural networks that deal with signals defined over graphs. As such, Graph Convolutional Recurrent Network (GCRN) is an evolved version of GNN that is a generalisation of classical RNN to data structured in a graph format, and can be defined as a deep learning model capable of predicting structured sequences of data. For computational efficiency, GCRN combines Convolutional Neural Networks (CNN) on graphs to identify spatial structures, and RNN to find dynamic patterns [110].

Consider $x_t \in \mathscr{D}$ to be an observation at time t, where $\mathscr{D}$ denotes the domain of observed features. The Clearwater vIMS architecture as defined in Figure 5.2 can be defined as an undirected and unweighted graph $\mathscr{G} = (\mathscr{V}, \mathscr{E}, A)$ as shown in Figure 5.3, where $\mathscr{V}$ is a finite set of $|\mathscr{V}| = n = 6$ vertices denoting the 6 Clearwater VNFCs, $\mathscr{E}$ as the set of edges, and $A \in \mathbb{R}^{n \times n}$ as the adjacency matrix denoting the (optional) weight of connection between two vertices. Therefore, a signal $x_t : \mathscr{V} \longrightarrow \mathbb{R}^{d_x}$ defined on the nodes of the Clearwater graph may be regarded as a matrix $x_t \in \mathbb{R}^{n \times d_x}$, whose column i is the d_x-dimensional value of x_t at the i^{th} node.

Figure 5.3 provides an overview of the developed forecasting model. We define this Clearwater GCRN model with GRU as the RNN component, i.e. as a Chebyshev Graph Convolutional Gated Recurrent Unit Cell (GConvGRU) with Chebyshev filter [110] of size K. K here controls the communication overhead, i.e. the number of nodes a given node i should exchange information with in order to compute its local states.

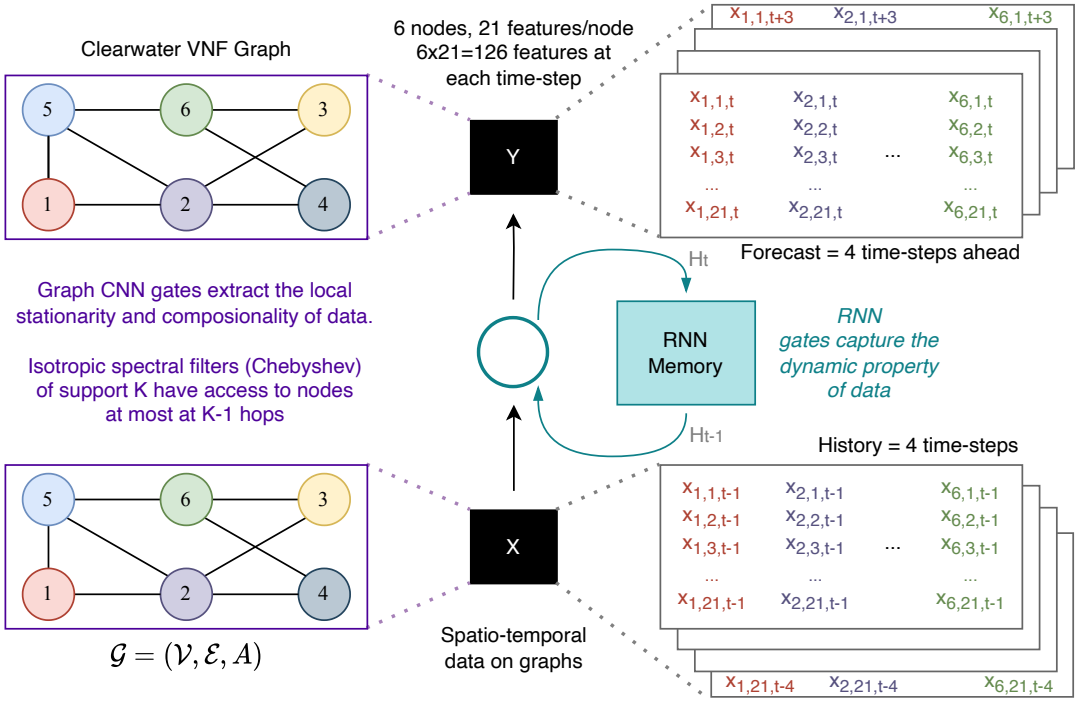


Fig. 5.3 Overview of the GCRN-based Clearwater feature forecasting model. GCRN is a special kind of GNN that merges CNN for graph-structured data and RNN to simultaneously identify meaningful spatial structures and dynamic patterns.

Mathematically, we model the forecasting problem such that we take a history of each of the Clearwater VNFC features for the past $H = 4$ time steps, and forecast the evolution of these features over the next $F = 4$ time steps:

$$\hat{Y}_t, ..., \hat{Y}_{t+F} = argmax_{Y_t,...,Y_{t+F}} P(Y_t, ..., Y_{t+F} | X_{t-H}, ..., X_{t-1}) \tag{5.6}$$

$X_t \in \mathscr{D}$ here is the matrix of the numeric state of each of the 21 features for the 6 VNFCs as observed at time t, $Y_t \in \mathscr{D}$ denotes the corresponding matrix of expected data features, and $\hat{Y}_t \in \mathscr{D}$ denotes the corresponding matrix of forecasted data features. P models the probability of expected features Y in a window of size F to appear conditioned on the past X features in a window of size H. Since we have spatial dependencies, the features of observations within X, Y, and $\hat{Y}$ are linked by pairwise relationships, modelled by $\mathscr{G}$.

Since we use a GRU based GCRN, the model (GConvGRU) can be represented as:

$$\begin{aligned} z &= \sigma(W_{xz} *_{\mathcal{G}} X_t + W_{hz} *_{\mathcal{G}} h_{t-1}), \\ r &= \sigma(W_{xr} *_{\mathcal{G}} X_t + W_{hr} *_{\mathcal{G}} h_{t-1}), \\ \tilde{h} &= tanh(W_{xh} *_{\mathcal{G}} X_t + W_{hh} *_{\mathcal{G}} (r \odot h_{t-1})) \\ h_t &= z \odot h_{t-1} + (1-z) \odot \tilde{h} \end{aligned} \tag{5.7}$$

z, r, $\tilde{h}$, and h_t are GRU parameters— z represents the update gate, r represents the reset gate, h_t represents the hidden state at time t, and $\tilde{h}$ represents the new hidden state. Further, $\odot$ represents the Hadamard product, and σ represents the logistic sigmoid function. $*_{\mathcal{G}}$ then represents the graph convolution operator, and the support K of the graph convolutional kernels defined by the the Chebyshev coefficients $W_{x\cdot}$ and $W_{h\cdot}$ determines the number of parameters independent of the number of nodes n.

Algorithm 5.1 presents the procedural workflow adopted towards achieving such a model that learns spatio-temporal structures from graph-structured and time-varying data. We evaluate this methodology in the following sections, benchmarking it against the current state-of-art on the use-case.

5.4.2 Deep Reinforcement Learning (DRL)

When it comes to dynamic application scenarios, RL is an effective machine learning methodology. A RL agent directly interacts with the environment to form a policy for decision-making based on a reward mechanism, that is customizable to achieve the desired outcomes. Specifically, RL follows the Markov Decision Process (MDP) model to train an agent that iteratively observes the state s_t of the environment at a discrete time step t to prescribe action a_t that maximizes the reward r_t. Cumulating and maximizing the expectation of rewards over time, RL thereby attains an efficient policy π for stochastic scenarios. Q-Learning in this regard is a typical off-policy model-free RL algorithm that calculates the value of each state-action pair as a Q-value function $Q(s,a)$:

$$Q(s_{t+1}, a_{t+1}) = \mathbb{E}\left[r_t + \gamma \, max_{a'} Q_i(s', a') | (s_t, a_t)\right] \tag{5.8}$$

Then, based on the policy $\pi = P(a|s)$ (e.g. ε-greedy), it chooses the action with the largest Q-value (Q^*), and follows the gradient towards higher rewards:

$$Q^*(s,a) = max_{\pi} \mathbb{E}\left[r_t + \gamma \, r_{t+1} + \gamma^2 r_{t+2} \cdots | s_t = s, a_t = a, \pi\right] \tag{5.9}$$

γ here is the discount factor applied to the rewards at each time step, accounting for the diminishing value of the reward at time t as we iterate forward. DRL uses deep neural

networks as non-linear function approximators to estimate the action-value function and deal with a large range of outcomes and their impact over time:

$$Q(s,a;\theta) \approx Q^*(s,a) \tag{5.10}$$

θ here refers to the weights of the neural network function approximator. Further, with the use of experience replay and network cloning, DRL is adept to deal with a complex environment with a big range of outcomes [111].

As stated previously, there is an active trade-off between efficiency and reliability in the existing scaling policies for latency-critical applications by service providers [16]. The aim of the reinforcement learning module here is to oversee and interface between an existing scaling policy and the defined SLA, and proactively generate warning alerts for anticipated SLA violations based on the GNN-based feature forecasting module and the defined SLOs. Building upon our existing work on granular SLA and SLO violation prediction[8], this serves as a proof-of-concept for dynamic SLA-aware policy adjustment.

For the deep Q-learning problem formulation, we consider the SLA violated (state 1) if any of the 4 SLOs as defined in Section 5.3 are violated, i.e.,

$$SLA = \begin{cases} 1, & \text{if any } SLO_j \text{ in } \mathscr{L} \text{ is } Violated \\ 0, & \text{otherwise} \end{cases} \tag{5.11}$$

Considering the information collected in the data we're working with, the action space is defined as:

$$a_t = \begin{cases} 1, & \text{i.e. } Scale\ up \\ 0, & \text{i.e. } No\ requirement\ to\ scale\ up \end{cases} \tag{5.12}$$

Correspondingly, we define the reward function as follows to cover all possible scenarios of the problem statement, and set the values as such so that there is a significant difference in optimal, sub-optimal, and non-optimal scenarios.

$$r_t = \begin{cases} +20, & \text{if } a_t = 1 \text{ and } SLA = 1 \\ -10, & \text{if } a_t = 0 \text{ and } SLA = 1 \\ -5, & \text{if } a_t = 1 \text{ and } SLA = 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.13}$$

Algorithm 5.1 details the workflow of the reinforcement learning model within the overall framework deployed, and Figure 5.4 presents its corresponding graphical overview.

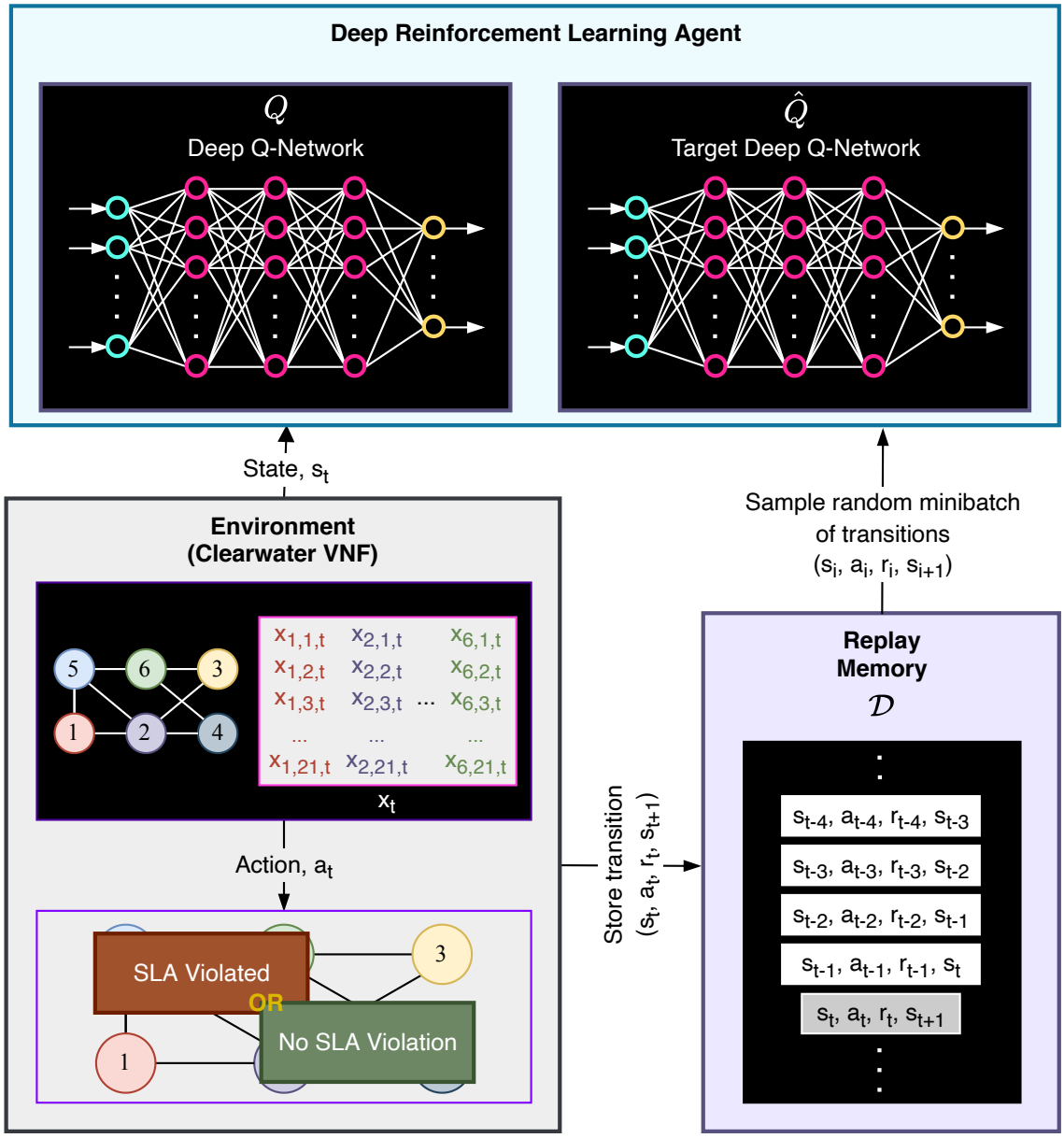


Fig. 5.4 Overview of the deep Q-learning based Clearwater DRL model.

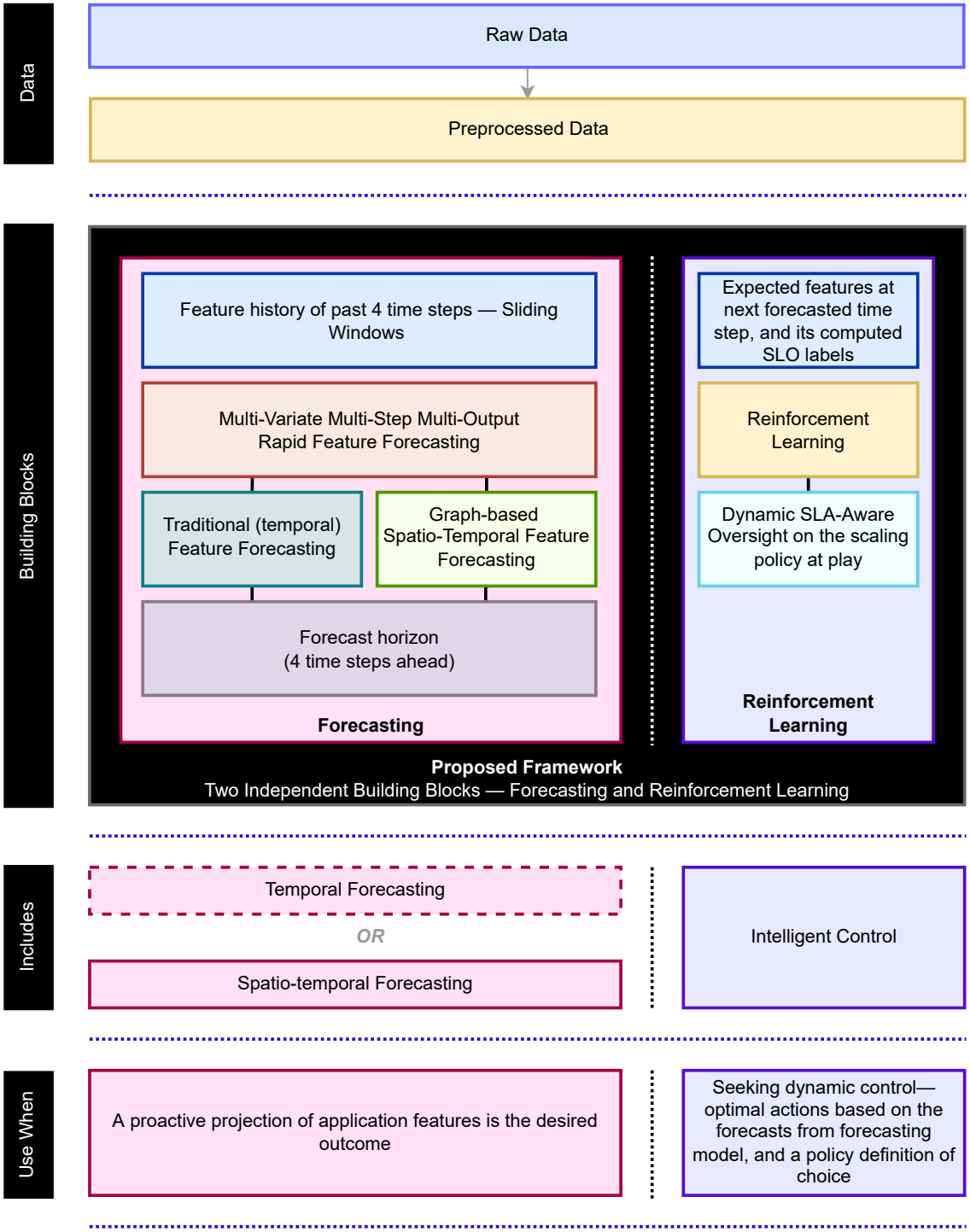


Fig. 5.5 A high-level overview of the framework's building blocks, and what they deliver.

Figure 5.5 provides another view at how GCRN and GRL work together within the framework, and what they deliver independently. Further, the following sections elaborate on the learning and adaptation for the developed framework.

5.5 Experimental Setup

The experiments were all set up using Python (version 3.9.7) and its associated data-science libraries. We use PyTorch[3] and PyTorch Lightning [4] to program all the deep learning based implementations in the Chapter. For the GCRN implementation, we use PyTorch Geometric Temporal [112], a temporal GNN extension library for PyTorch Geometric that supports the processing of spatio-temporal signals. We program a custom Gym [5] environment that supports Clearwater data for the DRL implementation, and use Stable Baselines3 [6] to implement deep Q-learning in PyTorch.

5.5.1 Dataset

We use a publicly hosted dataset[7] obtained via a standard Clearwater testbed, a visualization for which is as presented earlier in Figure 5.1. The dataset comprises of raw telemetric data files that track 26 metrics for each of the 6 monitored VNFs that compose the Clearwater ecosystem, and includes bursts of abnormal behaviour through its integrated stress testing tools to simulate VNF congestions and QoS degradations. The data is sampled every 30 seconds, and spans an overall period of 2 months. This corresponds to 156 features overall, and 177,098 rows of raw temporal data. A brief description of the captured metrics is summarised in Table 5.1. Outcomes of the exploratory data analysis as performed on this data-set have been presented in Appendix A.

5.5.2 System Configuration

The experiments were performed in a Docker[8] based containerized environment running atop a bare-metal Linux server with 64 GB RAM, Intel® Xeon® CPU E5-2660 v2 @ 2.20GHz (40 physical processors), 2 NVIDIA® Tesla K20m GPUs, and 500 GB local storage. The Docker image runs an Ubuntu 20.04 LTS operating system, and CUDA version 11.3 for the GPUs.

[3] www.pytorch.org
[4] www.pytorchlightning.ai
[5] www.gym.openai.com
[6] www.github.com/DLR-RM/stable-baselines3
[7] www.bit.ly/3gPY8c5
[8] www.docker.com

5.5.3 Learning and Adaptation

We adopt a sliding window methodology for time-series forecasting, which is suitable for rapid forecasting. We tested the models on different window sizes, and considering the short-term dynamics of the use-case, the input window size was optimally set to 4, and the models forecast the possibilities of SLO violations over each of the next 4 time-steps. Thus, since the sampling frequency is 30 second intervals, we use the data for all the features over the last 2 minutes to forecast the specific SLOs that may be violated at each step over the next 2 minutes.

We split the available data into training and test sets in the ratio of 80 : 20, and the training set is further split into training and validation sets in the ratio of 80 : 20. Hence, overall, the data consisting of 177,098 rows is split in the ratio 64 : 20 : 16, corresponding to train, test, and validation splits respectively. Further, given the data has a high degree of very large outliers due to the incorporated stress tests, and the scales vastly vary for each of the features depending on the category of raw metric, we use a non-linear transformation method to transform the very skewed nature of this dataset and map it to a uniform distribution in $[0, 1]$. This pre-processing is done via a Quantile Transformer, and this makes it suitable for learning by neural network methodologies.

For training the forecasting model, we use Mean Squared Error (MSE) as the loss function to be minimized. Further, we use Adam as the optimizer for its adaptive nature, with a learning rate of 10^{-2}. ReLu (Rectified Linear Unit) is used as the activation function between layers due to its computational simplicity and high optimization performance [100]. For the general layers, we use the default PyTorch initializations for weights, activation, and bias. To control overfitting as the model gains complexity, we deploy an early stopping criteria that tracks the validation MSE with a patience of 10 epochs to ensure that the training is not stopped at a local optimization minima. At the end of training, model weights are restored from the best epoch, which is considered as the best performance achieved during training, before the model began to overfit on the training set.

For the deep Q-learning model, the observation space is defined as a discrete snapshot of the forecasted features at time t, i.e., the episode length is defined as 1. We use a multi-layer perceptron based deep-learning policy for the Q-value approximation and the target network. We train the model for 5,000,000 timesteps, set the learning rate to 0.0001, and stick to default values from original algorithm propositions in [111] and [113] for most of the hyperparameters. To elaborate, the size of the replay buffer was set to 1,000,000, the minibatch size for each gradient update was set at 32, the update coefficient τ was set to hard update at 1, the discount factor γ was set to 0.99. The model collects transitions for 50,000 timesteps before learning starts, and the model is set to update every

4 steps, with 1 gradient step after each rollout. The replay buffer class was set to automatic selection, and the target network $\hat{Q}$ was set to update every 10,000 environment steps, with the maximum value for the gradient clipping was set to 10. The initial value of the random action probability ε was set to 1, while the final value of ε was set to 0.05, with the exploration fraction set to 10% of the training period.

5.6 Results and Discussion

Table 5.2 Performance of two GCRN model architectures on forecasting metrics.

Model	In Channels	Out Channels	Filter Size	MSE	MAE	RMSE
GCRN	4	128	K = 3	0.036	0.132	0.188
GCRN	4	2048	K = 3	**0.034**	**0.128**	**0.185**

Table 5.2 presents the results for the performance of two architecture variants of the GCRN model on forecasting metrics. To set this in perspective, Figure 5.6 compares the mean absolute error (MAE) of various models that have been previously applied to the use-case data [9] with the same forecasting horizon. Our proposed GCRN-based spatio-temporal forecasting model achieves a 74.62% improvement over the Residual-LSTM [9] model that was presented as the best-in-class among the state-of-the-art in Chapter 4.

Further, Figures 5.7 and 5.8 demonstrate good convergence of the proposed deep Q-learning model while training, with incremental improvements in learned policy. Figure 5.9 shows the results when the trained model policy was tested on a random test sample of 100 episodes, delivering a positive reward 98% of the time.

5.7 Summary and Conclusion

In this Chapter, we propose a GNN and DRL based framework for proactive SLA management in the use-case of Clearwater, a latency-critical NFV application. We compose a Graph Convolutional Recurrent Network (GCRN) based spatio-temporal multivariate time-series forecasting model that forecasts the evolution of system monitoring features for the Clearwater VNF over the next 4 time steps, delivering 74.62% improved performance over the established baseline state-of-art model on the use-case. The wide leap in forecasting performance with our proposed framework quantifies the benefits of incorporating spatial metadata in use-cases that require multi-dimensional feature forecasting in high-frequency temporal data flows. Further, we leverage realistic Clearwater SLA and SLO definitions to

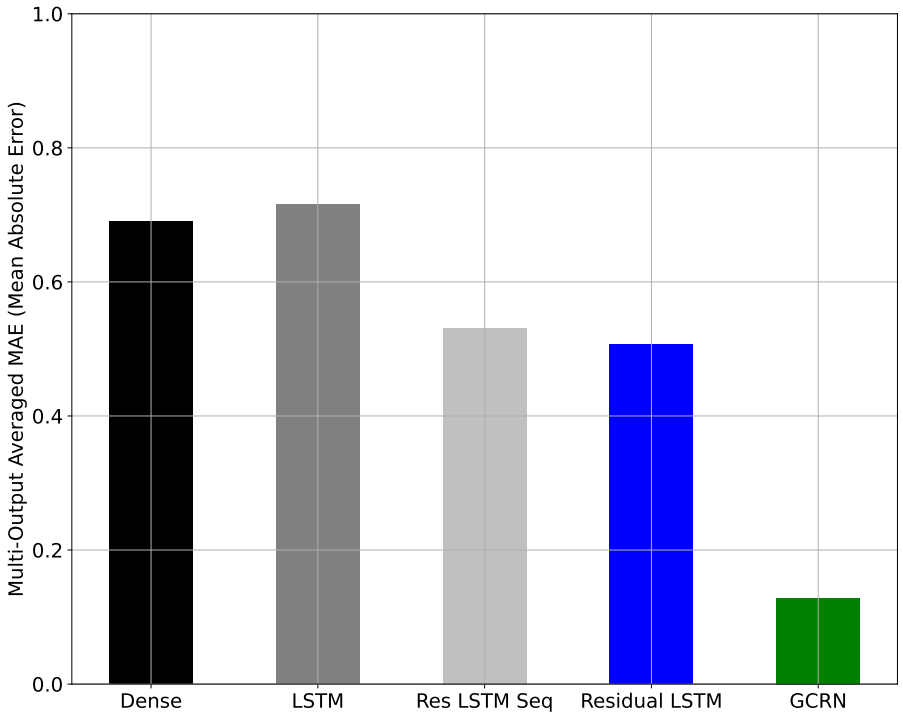


Fig. 5.6 Multi-output forecasting performance (MAE) of proposed Clearwater GCRN model against the previously established state-of-art models on the use-case, all configured with the same 2048-dimensional wide model architecture.

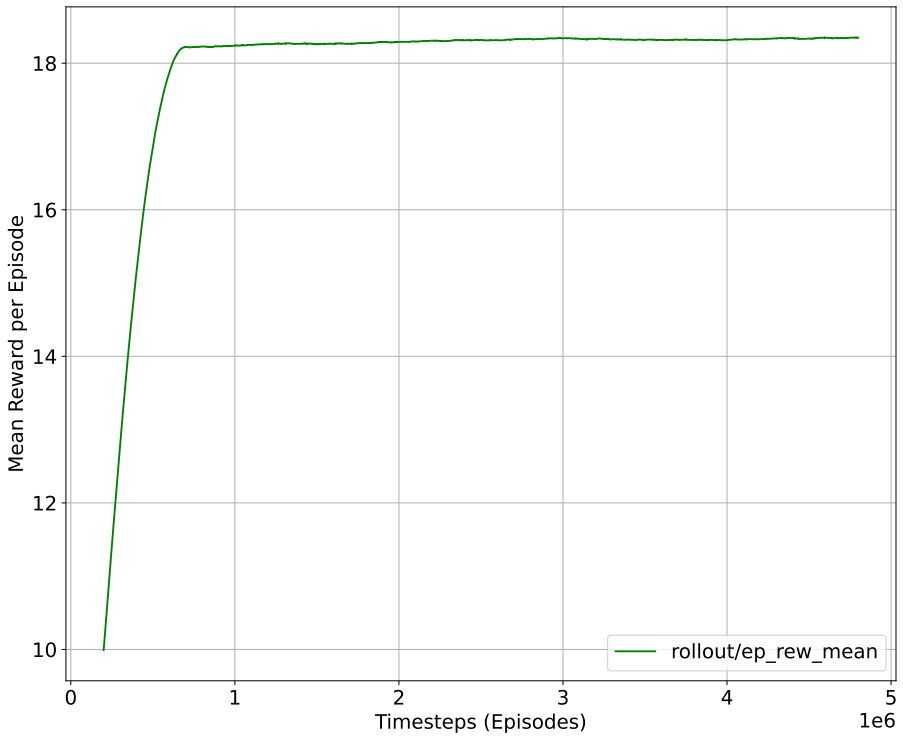


Fig. 5.7 Mean episodic DRL training reward (averaged over 100 episodes, further smoothed with a rolling mean of 100,000 timesteps for a better visualization of trend).

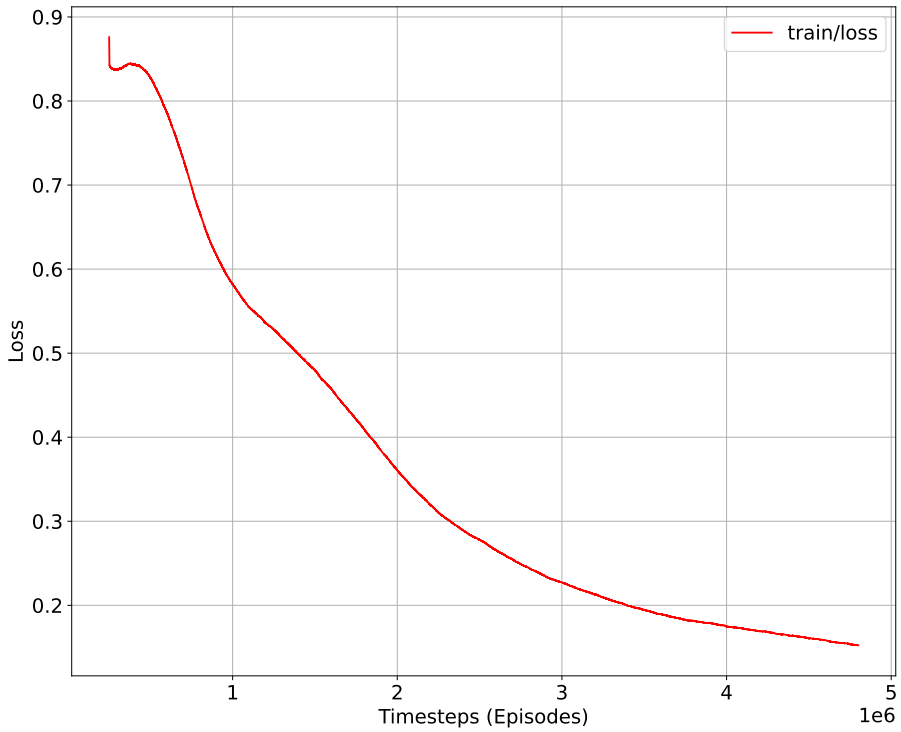


Fig. 5.8 Total loss value observed at each timestep as DRL training progresses (smoothed with a rolling mean of 100,000 timesteps for a better visualization of trend).

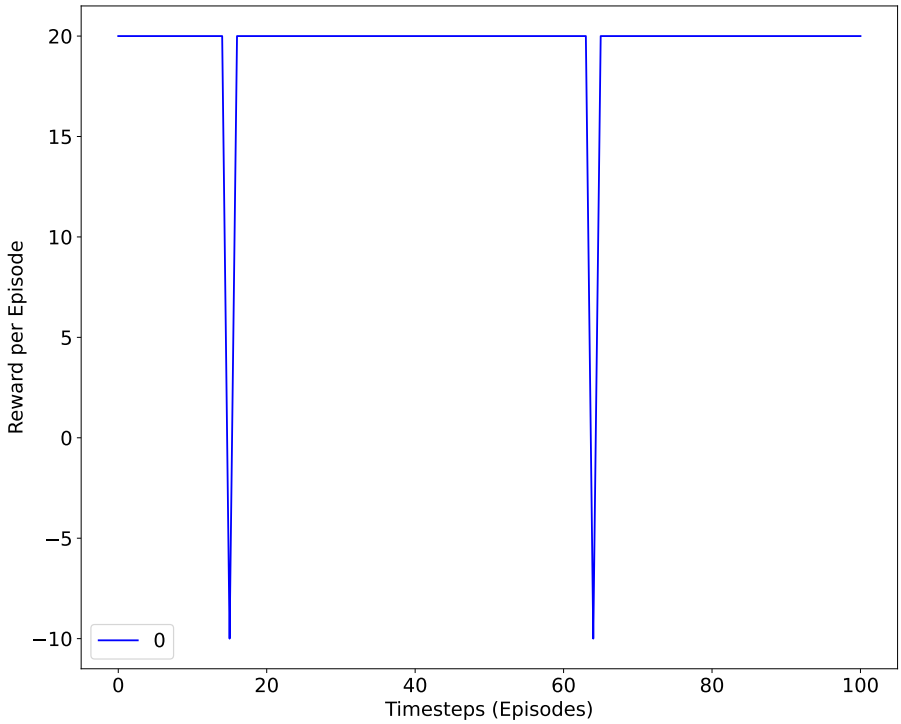


Fig. 5.9 Reward observed per episode when testing the trained deep Q-learning model.

develop a deep Q-learning based DRL model, and train it to act as an effective SLA-aware dynamic oversight of the scaling policy at play.

Chapter 6

Conclusions and Future Work

This chapter summarises our work presented in the dissertation, and maps the direction for future work. The main objective of this dissertation was to design and develop algorithms to address the complexity towards meeting QoS demands in serving upcoming verticals through the softwarised network architecture, and develop deep learning based frameworks for proactive SLA management in the use-case of a latency-critical NFV application. Working with real-world data from a latency sensitive NFV application, we established realistic multi-output models towards SLA and SLO violation prediction. Such modelling enables us to gain granular SLA and SLO violation insights, and presents an opportunity to study and mitigate their impact and inform precision in drafting proactive scaling policies in future work. Figure 6.1 presents an overview of the domain, the contributions made, and summarises the utility of each of the model propositions. The dissertation highlights the tremendous impact that various categories of deep learning based models hold in adapting to the increasing complexity of next-generation application requirements in latency-sensitive softwarised systems.

The previous chapters of the dissertation have discussed each of the research questions in detail, providing insights into the foundational research work while mapping contributions to the core areas of proactive SLA management with deep learning. This chapter focuses on concluding the work presented, and a discussion of possible avenues for further research. We provide a short summary of the chapters in §6.1, and the various directions for future work in §6.2.

6.1 Summary

Recent advancements in the domain of Network Function Virtualization (NFV), and rollout of next-generation networks have necessitated the requirement for the upkeep of latency-critical application architectures in future networks and communications. While

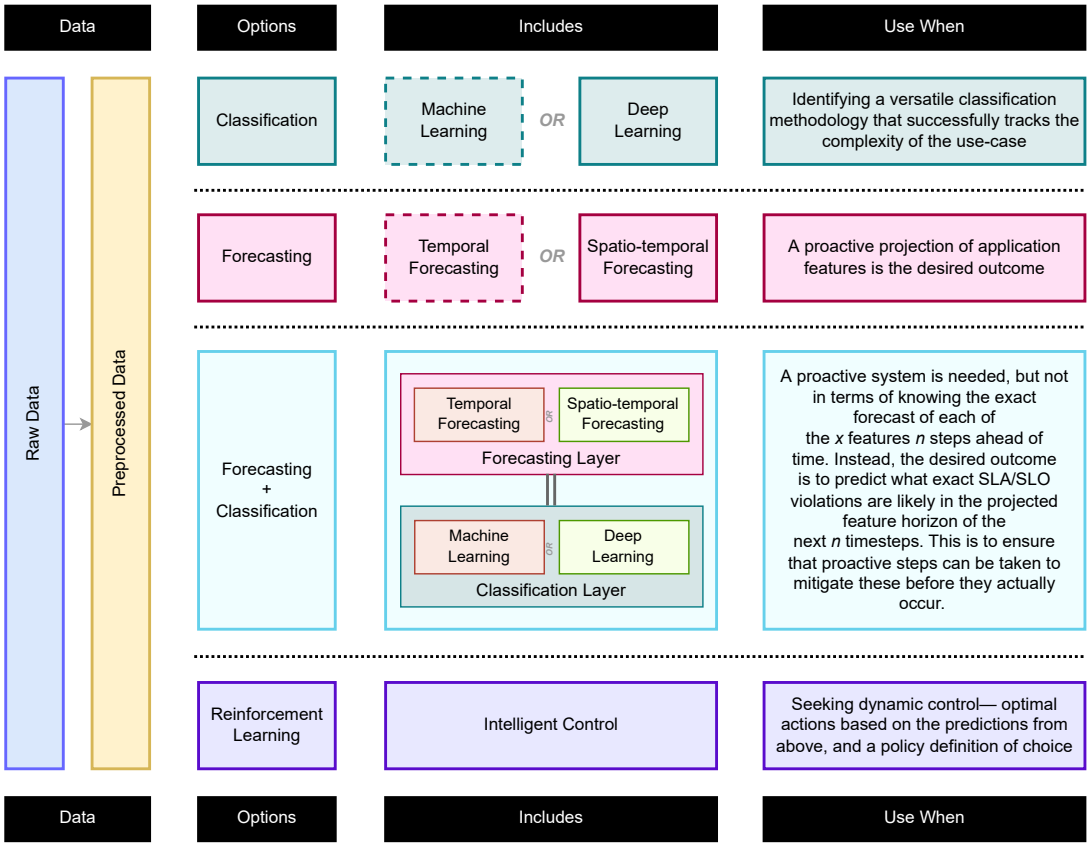


Fig. 6.1 An overview of the domain, the contributions made, and a summary of the utility of each of our model propositions.

Cloud service providers recognize the evolving mission-critical requirements in latency sensitive verticals such as autonomous driving, multimedia, gaming, telecommunications, and virtual reality, there is a wide gap to bridge the Quality of Service (QoS) constraints for the end-user experience. Most latency-critical services are over-provisioned on all fronts to offer reliability, which is inefficient towards scalability in the long run.

6.1.1 Chapter 3

In chapter 3, we address the above by proposing a strategy to model frequent violations on the application level as a multi-output target to enable more complex decision-making in the management of virtualised communication networks. We utilize data from a real-world deployment to configure and draft a realistic set of Service Level Objectives (SLOs) for a voice based NFV application, and propose the use of a deep neural network based multi-label classification methodology to identify and predict multiple categories of SLO breaches associated with an application state. With this, we aim to gain granular SLA and SLO violation insights, enabling us to study and mitigate their impact and inform precision

in drafting proactive scaling policies. We further compare the performance against a set of multi-label compatible machine learning classifiers, and address class imbalance in a multi-label setup. We perform a comprehensive evaluation to assess the performance on example-based, label-based and ranking-based measures, and demonstrate the suitability of deep learning in such a use-case.

6.1.2 Chapter 4

In chapter 4, we present a Residual Long Short-Term Memory (LSTM) based multi-label classification framework for proactive SLA management in the application use case. We compose a multivariate time-series forecasting model with multiple time-step predictions in a multi-output scenario, and associate a multi-label classifier for a granular prediction of individual Service Level Objective (SLO) violations for each step in the forecast horizon. We demonstrate the suitability of the Residual LSTM model over other MLP and LSTM based methodologies in such a scenario that involves fine-grained rapid forecasting, and reason that the high level of granularity in predicting SLOs as multi-label outputs would help ensure a balance in precise provisioning while maintaining reliability in latency-critical NFV applications.

6.1.3 Chapter 5

In chapter 5, we look to tackle the over-provisioning of latency-critical services by proposing a proactive SLA management framework leveraging Graph Neural Networks (GNN) and Deep Reinforcement Learning (DRL) to balance the trade-off between efficiency and reliability. We compose a Graph Convolutional Recurrent Network (GCRN) based spatio-temporal multivariate time-series forecasting model with multiple time-step predictions in a multi-output scenario, delivering a wide leap in performance over the established baseline state-of-art model on the use-case. This quantifies the benefits of incorporating spatial metadata in use-cases that require multi-dimensional feature forecasting in high-frequency temporal data flows. Further, we leverage realistic Clearwater SLA and SLO definitions to develop a deep Q-learning based DRL model, and train it to act as an effective SLA-aware dynamic oversight of the scaling policy at play.

6.2 General Limitations, and Future Work

As with any study, there is always an associated set of limitations, and a scope for future work. One of the key contributions of the research as presented in this dissertation is that the underlying real-world data(set) and the custom defined SLA/SLO labels have been

kept constant throughout all the stages of the work. This presents a stable benchmarking within each of the target areas, and makes all competing methodologies directly and quantifiably comparable— something that was lacking in the state-of-art thus far. However, this also brings in some limitations in terms of factoring in external variables while increasing the scope of the research. The most prominent avenue of future work here is to go beyond application data— to set up a wider testbed with both SDN and NFV components interlinked, and extend the collected and incorporated features to include the external variables like miscellaneous network features, and factor in distributed application scenarios. Further, since we have used general server metrics to draft the SLA and SLO definitions, the methodology is transferable to other verticals within the high-availability network slice. There is a good scope to validate this on a different vertical, and compare the results achieved.

The testbed would also ensure an even closer environment to that of production systems, with opportunity to include frequent re-training loops that address the ever-evolving data patterns. As methodologies and principles evolve across systems and domains, and new principles like site reliability engineering take over the legacy operational practices, machine learning has a key role to play. The next generation of systems contain requirements that move beyond traditional monitoring, now venturing into the domain of observability. Proactive problem and event management is the need of the hour to support Cloud-based infrastructure and services for next generation verticals. To this end, one of the key challenges in production systems is to tackle data drift— models trained offline and then deployed online are effective only as long as the data distribution remains equivalent to what the model has been trained on, and the features' behaviour lies within the sample subset of scenarios that the training set contains. To tackle this, it is important to configure retraining feedback loops that periodically expose the model to a new subset of incoming data offline, allow a readjustment of hyperparameters inline with the new observations if any, and then a redeployment of the updated model in production. Retraining overheads are a key consideration in production deployments, and something that could become a potential demerit of an otherwise well-performing model. Moreover, an enhanced ability to react to the unknown is another reason why reinforcement learning is an important layer of oversight over the supervised learning models. Having a live testbed would also be helpful to improve the reinforcement learning model for extended online sequence modeling with longer episode lengths.

Another direction of future work is to study the implications of the proposed approaches in this dissertation to directly improve the existing application scaling policy within the vertical in a control loop. Further, while the data we worked with includes bursts of abnormal behaviour through its integrated stress testing tools to simulate VNF congestions

and QoS degradations, the proposed approaches from this dissertation could also be integrated with a traffic and workload forecasting methodology for a higher degree of detail in proactive violations' prediction, and then combined with a dynamic policy enforcement for a wider end-to-end management control loop.

Bibliography

[1] ITU, "IMT traffic estimates for the years 2020 to 2030, ITU-R M.2370-0," International Telecommunication Union, Geneva, Switzerland, Tech. Rep., Jul. 2015, Available: https://bit.ly/3g828Ux. [Online]. Available: https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-M.2370-2015-PDF-E.pdf (visited on 05/31/2021).

[2] Ericsson, "Mobility Report," Stockholm, Sweden, Tech. Rep., Nov. 2020, Available: https://bit.ly/3gmcI9E. [Online]. Available: https://www.ericsson.com/4adc87/assets/local/mobility-report/documents/2020/november-2020-ericsson-mobility-report.pdf (visited on 05/31/2021).

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016, ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2477041. [Online]. Available: http://ieeexplore.ieee.org/document/7243304/ (visited on 04/20/2021).

[4] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The Road Towards 6G: A Comprehensive Survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021, ISSN: 2644-125X. DOI: 10.1109/OJCOMS.2021.3057679. [Online]. Available: https://ieeexplore.ieee.org/document/9349624/ (visited on 04/20/2021).

[5] T. Zhang, H. Qiu, L. Linguaglossa, W. Cerroni, and P. Giaccone, "NFV Platforms: Taxonomy, Design Choices and Future Challenges," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 30–48, Mar. 2021, ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2020.3045381. [Online]. Available: https://ieeexplore.ieee.org/document/9302614/ (visited on 04/20/2021).

[6] M. Di Mauro, G. Galatro, M. Longo, F. Postiglione, and M. Tambasco, "IP Multimedia Subsystem in a containerized environment: Availability and sensitivity evaluation," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, Paris, France: IEEE, Jun. 2019, pp. 42–47, ISBN: 978-1-5386-9376-6. DOI: 10.1109/NETSOFT.2019.8806642. [Online]. Available: https://ieeexplore.ieee.org/document/8806642/ (visited on 04/26/2021).

[7] N. Jalodia, S. Henna, and A. Davy, "Deep Reinforcement Learning for Topology-Aware VNF Resource Prediction in NFV Environments," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Dallas, TX, USA: IEEE, Nov. 2019, pp. 1–5, ISBN: 978-1-72814-545-7. DOI: 10.1109/NFV-SDN47374.2019.9040154. [Online]. Available: https://ieeexplore.ieee.org/document/9040154/ (visited on 01/31/2021).

[8] N. Jalodia, M. Taneja, and A. Davy, "A Deep Neural Network-Based Multi-Label Classifier for SLA Violation Prediction in a Latency Sensitive NFV Application," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 2469–2493, 2021, ISSN: 2644-125X. DOI: 10.1109/OJCOMS.2021.3122844. [Online]. Available: https://ieeexplore.ieee.org/document/9592636/ (visited on 03/18/2022).

[9] N. Jalodia, M. Taneja, A. Davy, and B. Dezfouli, "A Residual LSTM based Multi-Label Classification Framework for Proactive SLA Management in a Latency Critical NFV Application Use-Case," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA: IEEE, Jan. 2022, pp. 782–789, ISBN: 978-1-66543-161-3. DOI: 10.1109/CCNC49033.2022.9700502. [Online]. Available: https://ieeexplore.ieee.org/document/9700502/ (visited on 03/18/2022).

[10] B. Beyer, C. Jones, J. Petoff, and N. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Incorporated, 2016, ISBN: 978-1-4919-2912-4. [Online]. Available: https://books.google.ie/books?id=81UrjwEACAAJ.

[11] *Dynamic scaling for Amazon EC2 Auto Scaling - Amazon Web Services (AWS)*, https://amzn.to/3yPU963, accessed May 2021. [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html (visited on 05/31/2021).

[12] *Predictive scaling for Amazon EC2 Auto Scaling - Amazon Web Services (AWS)*, https://amzn.to/3c5pQPc, accessed May 2021. [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-predictive-scaling.html (visited on 05/31/2021).

[13] *Predictive Scaling for EC2 with Machine Learning - Amazon Web Services (AWS)*, https://amzn.to/3vBHlyg, accessed May 2021. [Online]. Available: https://aws.amazon.com/blogs/aws/new-predictive-scaling-for-ec2-powered-by-machine-learning/ (visited on 05/31/2021).

[14] *Using Predictive Autoscaling - Google Cloud*, https://bit.ly/3vFc5hH, accessed May 2021. [Online]. Available: https://cloud.google.com/compute/docs/autoscaler/predictive-autoscaling (visited on 05/31/2021).

[15] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of Network Function Virtualization," en, *Computer Networks*, vol. 133, pp. 212–262, Mar. 2018, ISSN: 13891286. DOI: 10.1016/j.comnet.2018.01.021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128618300306 (visited on 04/20/2021).

[16] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-Aware Prediction of Virtual Network Function Resource Requirements," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 106–120, Mar. 2017, ISSN: 1932-4537. DOI: 10.1109/TNSM.2017.2666781. [Online]. Available: http://ieeexplore.ieee.org/document/7849149/ (visited on 02/04/2020).

[17] S. Cherrared, S. Imadali, E. Fabre, and G. Goessler, "LUMEN: A global fault management framework for network virtualization environments," in *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, Paris: IEEE, Feb. 2018, pp. 1–8, ISBN: 978-1-5386-3458-5. DOI: 10.1109/ICIN.2018.8401622. [Online]. Available: https://ieeexplore.ieee.org/document/8401622/ (visited on 04/20/2021).

[18] A. Binsahaq, T. R. Sheltami, and K. Salah, "A Survey on Autonomic Provisioning and Management of QoS in SDN Networks," *IEEE Access*, vol. 7, pp. 73 384–73 435, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2919957. [Online]. Available: https://ieeexplore.ieee.org/document/8726285/ (visited on 04/20/2021).

[19] S. Cherrared, S. Imadali, E. Fabre, G. Gossler, and I. G. B. Yahia, "A Survey of Fault Management in Network Virtualization Environments: Challenges and Solutions," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1537–1551, Dec. 2019, ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2019.2948420. [Online]. Available: https://ieeexplore.ieee.org/document/8877749/ (visited on 04/20/2021).

[20] X. Zheng, N. Huang, S. Yin, G. Wen, and X. Zhang, "A Service Deployment Method Considering Application Reliability of Networks," *IEEE Access*, vol. 9, pp. 28 505–28 513, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3053961. [Online]. Available: https://ieeexplore.ieee.org/document/9334977/ (visited on 04/26/2021).

[21] J. Suomalainen, A. Juhola, S. Shahabuddin, A. Mammela, and I. Ahmad, "Machine Learning Threatens 5G Security," *IEEE Access*, vol. 8, pp. 190 822–190 842, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3031966. [Online]. Available: https://ieeexplore.ieee.org/document/9229146/ (visited on 04/20/2021).

[22] B. Tola, G. Nencioni, and B. E. Helvik, "Network-Aware Availability Modeling of an End-to-End NFV-Enabled Service," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1389–1403, Dec. 2019, ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2019.2948725. [Online]. Available: https://ieeexplore.ieee.org/document/8879511/ (visited on 04/26/2021).

[23] M. Di Mauro, M. Longo, F. Postiglione, G. Carullo, and M. Tambasco, "Service function chaining deployed in an NFV environment: An availability modeling," in *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, Helsinki, Finland: IEEE, Sep. 2017, pp. 42–47, ISBN: 978-1-5386-3070-9. DOI: 10.1109/CSCN.2017.8088596. [Online]. Available: http://ieeexplore.ieee.org/document/8088596/ (visited on 04/26/2021).

[24] J. Hong, S. Park, J.-H. Yoo, and J. W.-K. Hong, "Machine Learning based SLA-Aware VNF Anomaly Detection for Virtual Network Management," in *2020 16th International Conference on Network and Service Management (CNSM)*, Izmir, Turkey: IEEE, Nov. 2020, pp. 1–7, ISBN: 978-3-903176-31-7. DOI: 10.23919/CNSM50824.2020.9269100. [Online]. Available: https://ieeexplore.ieee.org/document/9269100/ (visited on 11/03/2021).

[25] D. de Vleeschauwer *et al.*, "5Growth Data-Driven AI-Based Scaling," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, Porto, Portugal: IEEE, Jun. 2021, pp. 383–388, ISBN: 978-1-66541-526-2. DOI: 10.1109/EuCNC/6GSummit51104.2021.9482476. [Online]. Available: https://ieeexplore.ieee.org/document/9482476/ (visited on 11/01/2021).

[26] D. Lee, J.-H. Yoo, and J. W.-K. Hong, "Deep Q-Networks based Auto-scaling for Service Function Chaining," in *2020 16th International Conference on Network and Service Management (CNSM)*, Izmir, Turkey: IEEE, Nov. 2020, pp. 1–9, ISBN: 978-3-903176-31-7. DOI: 10.23919/CNSM50824.2020.9269107. [Online]. Available: https://ieeexplore.ieee.org/document/9269107/ (visited on 11/03/2021).

[27] Y. Jiang, M. Kodialam, T. V. Lakshman, S. Mukherjee, and L. Tassiulas, "Fast reinforcement learning algorithms for resource allocation in data centers," in *2020 IFIP networking conference (networking)*, 2020, pp. 271–279.

[28] L.-V. Le, D. Sinh, B.-S. P. Lin, and L.-P. Tung, "Applying Big Data, Machine Learning, and SDN/NFV to 5G Traffic Clustering, Forecasting, and Management," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, Montreal, QC: IEEE, Jun. 2018, pp. 168–176, ISBN: 978-1-5386-4633-5. DOI: 10.1109/NETSOFT.2018.8460129. [Online]. Available: https://ieeexplore.ieee.org/document/8460129/ (visited on 04/20/2021).

[29] A. A. Gebremariam, M. Usman, and M. Qaraqe, "Applications of Artificial Intelligence and Machine Learning in the Area of SDN and NFV: A Survey," in *2019 16th International Multi-Conference on Systems, Signals & Devices (SSD)*, Istanbul, Turkey: IEEE, Mar. 2019, pp. 545–549, ISBN: 978-1-72811-820-8. DOI: 10.1109/SSD.2019.8893244. [Online]. Available: https://ieeexplore.ieee.org/document/8893244/ (visited on 04/20/2021).

[30] J. Vergara-Reyes, M. C. Martinez-Ordonez, A. Ordonez, and O. M. Caicedo Rendon, "IP traffic classification in NFV: A benchmarking of supervised Machine Learning algorithms," in *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*, Cartagena: IEEE, Aug. 2017, pp. 1–6, ISBN: 978-1-5386-1060-2. DOI: 10.1109/ColComCon.2017.8088199. [Online]. Available: https://ieeexplore.ieee.org/document/8088199/ (visited on 04/20/2021).

[31] G. Ilievski and P. Latkoski, "Efficiency of Supervised Machine Learning Algorithms in Regular and Encrypted VoIP Classification within NFV Environment," en, *Radioengineering*, vol. 29, no. 1, pp. 243–250, Apr. 2020, ISSN: 1210-2512. DOI: 10.13164/re.2020.0243. [Online]. Available: https://www.radioeng.cz/fulltexts/2020/20_01_0243_0250.pdf (visited on 04/20/2021).

[32] D. Ferreira, A. Braga Reis, C. Senna, and S. Sargento, "A Forecasting Approach to Improve Control and Management for 5G Networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1817–1831, Jun. 2021, ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2021.3056222. [Online]. Available: https://ieeexplore.ieee.org/document/9344586/ (visited on 07/31/2021).

[33] J. Bendriss, I. G. Ben Yahia, and D. Zeghlache, "Forecasting and anticipating SLO breaches in programmable networks," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, Paris: IEEE, Mar. 2017, pp. 127–134, ISBN: 978-1-5090-3672-1. DOI: 10.1109/ICIN.2017.7899402. [Online]. Available: http://ieeexplore.ieee.org/document/7899402/ (visited on 01/31/2021).

[34] M. Abbasi, A. Shahraki, and A. Taherkordi, "Deep Learning for Network Traffic Monitoring and Analysis (NTMA): A Survey," en, *Computer Communications*, vol. 170, pp. 19–41, Mar. 2021, ISSN: 01403664. DOI: 10.1016/j.comcom.2021.01.021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0140366421000426 (visited on 07/31/2021).

[35] C. Sun, J. Bi, Z. Zheng, and H. Hu, "SLA-NFV: An SLA-aware High Performance Framework for Network Function Virtualization," en, in *Proceedings of the 2016 ACM SIGCOMM Conference*, Florianopolis Brazil: ACM, Aug. 2016, pp. 581–582, ISBN: 978-1-4503-4193-6. DOI: 10.1145/2934872.2959058. [Online]. Available: https://dl.acm.org/doi/10.1145/2934872.2959058 (visited on 04/21/2021).

[36] E. Kapassa, M. Touloupou, and D. Kyriazis, "SLAs in 5G: A Complete Framework Facilitating VNF- and NS- Tailored SLAs Management," in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Krakow: IEEE, May 2018, pp. 469–474, ISBN: 978-1-5386-5395-1. DOI: 10.1109/WAINA.2018.00130. [Online]. Available: https://ieeexplore.ieee.org/document/8418115/ (visited on 04/20/2021).

[37] I. G. Ben Yahia, J. Bendriss, A. Samba, and P. Dooze, "CogNitive 5G networks: Comprehensive operator use cases with machine learning for management operations," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, Paris: IEEE, Mar. 2017, pp. 252–259, ISBN: 978-1-5090-3672-1. DOI: 10.1109/ICIN.2017.7899421. [Online]. Available: http://ieeexplore.ieee.org/document/7899421/ (visited on 04/20/2021).

[38] J. Bendriss, I. G. Ben Yahia, P. Chemouil, and D. Zeghlache, "AI for SLA Management in Programmable Networks," in *DRCN 2017 - Design of Reliable Communication Networks; 13th International Conference*, 2017, pp. 1–8.

[39] X.Li *et al.*, "Automated Service Provisioning and Hierarchical SLA Management in 5G Systems," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021, ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2021.3102890. [Online]. Available: https://ieeexplore.ieee.org/document/9508404/ (visited on 11/01/2021).

[40] M. Boucadair, C. Jacquenet, and X. Xu, Eds., *Emerging Automation Techniques for the Future Internet:* ser. Advances in Wireless Technologies and Telecommunication. IGI Global, 2019, ISBN: 978-1-5225-7146-9 978-1-5225-7147-6. DOI: 10.4018/978-1-5225-7146-9. [Online]. Available: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-5225-7146-9 (visited on 06/04/2021).

[41] J. Bendriss, "Cognitive management of SLA in software-based networks," Issue: 2018E0003, Theses, Institut National des Télécommunications, Jun. 2018. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01891046.

[42] J. Bendriss, I. G. Ben Yahia, and D. Zeghlache, "Forecasting and anticipating SLO breaches in programmable networks," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, Paris: IEEE, Mar. 2017, pp. 127–134, ISBN: 978-1-5090-3672-1. DOI: 10.1109/ICIN.2017.7899402. [Online]. Available: http://ieeexplore.ieee.org/document/7899402/ (visited on 04/20/2021).

[43] D. Xu, Y. Shi, I. W. Tsang, Y.-S. Ong, C. Gong, and X. Shen, "Survey on Multi-Output Learning," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2019, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2019.2945133. [Online]. Available: https://ieeexplore.ieee.org/document/8892612/ (visited on 04/30/2021).

[44] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining Multi-label Data," en, in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds., Boston, MA: Springer US, 2009, pp. 667–685, ISBN: 978-0-387-09822-7 978-0-387-09823-4. DOI: 10.1007/978-0-387-09823-4_34. [Online]. Available: http://link.springer.com/10.1007/978-0-387-09823-4_34 (visited on 05/07/2021).

[45] E. Gibaja and S. Ventura, "A Tutorial on Multilabel Learning," en, *ACM Computing Surveys*, vol. 47, no. 3, pp. 1–38, Apr. 2015, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/2716262. [Online]. Available: https://dl.acm.org/doi/10.1145/2716262 (visited on 05/06/2021).

[46] M. S. Sorrower, "A literature survey on algorithms for multi-label learning," pp. 1–25, 2010.

[47] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," en, *Pattern Recognition*, vol. 45, no. 9, pp. 3084–3104, Sep. 2012, ISSN: 00313203. DOI: 10.1016/j.patcog.2012.03.004. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0031320312001203 (visited on 05/01/2021).

[48] M.-L. Zhang and Z.-H. Zhou, "A Review on Multi-Label Learning Algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1819–1837, Aug. 2014, ISSN: 1041-4347. DOI: 10.1109/TKDE.2013.39. [Online]. Available: http://ieeexplore.ieee.org/document/6471714/ (visited on 04/30/2021).

[49] M. A. Tahir, J. Kittler, and F. Yan, "Inverse random under sampling for class imbalance problem and its application to multi-label classification," en, *Pattern Recognition*, vol. 45, no. 10, pp. 3738–3750, Oct. 2012, ISSN: 00313203. DOI: 10.1016/j.patcog.2012.03.014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0031320312001471 (visited on 05/02/2021).

[50] J. Shen, S. Li, F. Jia, H. Zuo, and J. Ma, "A Deep Multi-Label Learning Framework for the Intelligent Fault Diagnosis of Machines," *IEEE Access*, vol. 8, pp. 113 557–113 566, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3002826. [Online]. Available: https://ieeexplore.ieee.org/document/9118881/ (visited on 04/30/2021).

[51] W. Hong, W. Xu, J. Qi, and Y. Weng, "Neural Tensor Network for Multi- Label Classification," *IEEE Access*, vol. 7, pp. 96 936–96 941, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2930206. [Online]. Available: https://ieeexplore.ieee.org/document/8767910/ (visited on 04/30/2021).

[52] A. Maxwell *et al.*, "Deep learning architectures for multi-label classification of intelligent health risk prediction," en, *BMC Bioinformatics*, vol. 18, no. S14, p. 523, Dec. 2017, ISSN: 1471-2105. DOI: 10.1186/s12859-017-1898-z. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-017-1898-z (visited on 05/01/2021).

[53] J. Mandziuk and A. Zychowski, "Dimensionality Reduction in Multilabel Classification with Neural Networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*, Budapest, Hungary: IEEE, Jul. 2019, pp. 1–8, ISBN: 978-1-72811-985-4. DOI: 10.1109/IJCNN.2019.8852156. [Online]. Available: https://ieeexplore.ieee.org/document/8852156/ (visited on 05/01/2021).

[54] M. Ibrahim, M. Torki, and N. El-Makky, "Imbalanced Toxic Comments Classification Using Data Augmentation and Deep Learning," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Orlando, FL: IEEE, Dec. 2018, pp. 875–878, ISBN: 978-1-5386-6805-4. DOI: 10.1109/ICMLA.2018.00141. [Online]. Available: https://ieeexplore.ieee.org/document/8614166/ (visited on 05/01/2021).

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015, arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385 (visited on 07/31/2021).

[56] F. Wei, G. Feng, Y. Sun, Y. Wang, and S. Qin, "Proactive Network Slice Reconfiguration by Exploiting Prediction Interval and Robust Optimization," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, Taipei, Taiwan: IEEE, Dec. 2020, pp. 1–6, ISBN: 978-1-72818-298-8. DOI: 10.1109/GLOBECOM42002.2020.9322440. [Online]. Available: https://ieeexplore.ieee.org/document/9322440/ (visited on 07/31/2021).

[57] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009, ISSN: 1045-9227, 1941-0093. DOI: 10.1109/TNN.2008.2005605. [Online]. Available: http://ieeexplore.ieee.org/document/4700287/ (visited on 02/04/2020).

[58] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Jan. 2021, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2020.2978386. [Online]. Available: https://ieeexplore.ieee.org/document/9046288/ (visited on 06/15/2021).

[59] M. Ferriol-Galmés, J. Suárez-Varela, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Scaling Graph-based Deep Learning models to larger networks," *arXiv:2110.01261 [cs]*, Oct. 2021, arXiv: 2110.01261. [Online]. Available: http://arxiv.org/abs/2110.01261 (visited on 10/19/2021).

[60] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, "Graph Neural Networks for Scalable Radio Resource Management: Architecture Design and Theoretical Analysis," *arXiv:2007.07632 [cs, eess, math]*, Oct. 2020, arXiv: 2007.07632. [Online]. Available: http://arxiv.org/abs/2007.07632 (visited on 11/05/2021).

[61] O. Orhan, V. N. Swamy, T. Tetzlaff, M. Nassar, H. Nikopour, and S. Talwar, "Connection Management xAPP for O-RAN RIC: A Graph Neural Network and Reinforcement Learning Approach," *arXiv:2110.07525 [cs, math]*, Oct. 2021, arXiv: 2110.07525. [Online]. Available: http://arxiv.org/abs/2110.07525 (visited on 10/19/2021).

[62] S. Qi, S. Li, S. Lin, M. Y. Saidi, and K. Chen, "Energy-Efficient VNF Deployment for Graph-Structured SFC Based on Graph Neural Network and Constrained Deep Reinforcement Learning," in *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Tainan, Taiwan: IEEE, Sep. 2021, pp. 348–353, ISBN: 978-4-88552-332-8. DOI: 10.23919/APNOMS52696.2021.9562610. [Online]. Available: https://ieeexplore.ieee.org/document/9562610/ (visited on 10/19/2021).

[63] S. Yeom, C. Choi, S. S. Kolekar, and K. Kim, "Graph Convolutional Network based Link State Prediction," in *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Tainan, Taiwan: IEEE, Sep. 2021, pp. 246–249, ISBN: 978-4-88552-332-8. DOI: 10.23919/APNOMS52696.2021.9562682. [Online]. Available: https://ieeexplore.ieee.org/document/9562682/ (visited on 10/31/2021).

[64] Y. Cao, H. Jiang, Y. Deng, J. Wu, P. Zhou, and W. Luo, "Detecting and Mitigating DDoS Attacks in SDN Using Spatial-Temporal Graph Convolutional Network," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2021, ISSN: 1545-5971, 1941-0018, 2160-9209. DOI: 10.1109/TDSC.2021.3108782. [Online]. Available: https://ieeexplore.ieee.org/document/9527066/ (visited on 10/31/2021).

[65] A. Rafiq, T. A. Khan, M. Afaq, and W.-C. Song, "Service Function Chaining and Traffic Steering in SDN using Graph Neural Network," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea (South): IEEE, Oct. 2020, pp. 500–505, ISBN: 978-1-72816-758-9. DOI: 10.1109/ICTC49870.2020.9289378. [Online]. Available: https://ieeexplore.ieee.org/document/9289378/ (visited on 11/02/2021).

[66] H. Wang, Y. Wu, G. Min, and W. Miao, "A Graph Neural Network-based Digital Twin for Network Slicing Management," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2020, ISSN: 1551-3203, 1941-0050. DOI: 10.1109/TII.2020.3047843. [Online]. Available: https://ieeexplore.ieee.org/document/9310275/ (visited on 10/27/2021).

[67] H.-G. Kim *et al.*, "Graph Neural Network-based Virtual Network Function Deployment Prediction," in *2020 16th International Conference on Network and Service Management (CNSM)*, Izmir, Turkey: IEEE, Nov. 2020, pp. 1–7, ISBN: 978-3-903176-31-7. DOI: 10.23919/CNSM50824.2020.9269085. [Online]. Available: https://ieeexplore.ieee.org/document/9269085/ (visited on 11/03/2021).

[68] D. Heo, S. Lange, H.-G. Kim, and H. Choi, "Graph Neural Network based Service Function Chaining for Automatic Network Control," *arXiv:2009.05240 [cs]*, Sep. 2020, arXiv: 2009.05240. [Online]. Available: http://arxiv.org/abs/2009.05240 (visited on 11/05/2021).

[69] H. Zhu, V. Gupta, S. S. Ahuja, Y. Tian, Y. Zhang, and X. Jin, "Network planning with deep reinforcement learning," en, in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, Virtual Event USA: ACM, Aug. 2021, pp. 258–271, ISBN: 978-1-4503-8383-7. DOI: 10.1145/3452296.3472902. [Online]. Available: https://dl.acm.org/doi/10.1145/3452296.3472902 (visited on 10/31/2021).

[70] D. Yoon, S. Hong, B.-J. Lee, and K.-E. Kim, "Winning the L2{RPN} challenge: Power grid management via semi-markov afterstate actor-critic," in *International conference on learning representations*, 2021. [Online]. Available: https://openreview.net/forum?id=LmUJqB1Cz8.

[71] A. Swaminathan, M. Chaba, D. K. Sharma, and U. Ghosh, "GraphNET: Graph Neural Networks for routing optimization in Software Defined Networks," en, *Computer Communications*, vol. 178, pp. 169–182, Oct. 2021, ISSN: 01403664. DOI: 10.1016/j.comcom.2021.07.025. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0140366421002887 (visited on 11/01/2021).

[72] P. Sun *et al.*, "DeepMigration: Flow Migration for NFV with Graph-based Deep Reinforcement Learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Dublin, Ireland: IEEE, Jun. 2020, pp. 1–6, ISBN: 978-1-72815-089-5. DOI: 10.1109/ICC40277.2020.9148696. [Online]. Available: https://ieeexplore.ieee.org/document/9148696/ (visited on 10/28/2021).

[73] P. Sun, J. Lan, J. Li, Z. Guo, Y. Hu, and T. Hu, "Efficient flow migration for NFV with Graph-aware deep reinforcement learning," en, *Computer Networks*, vol. 183, p. 107 575, Dec. 2020, ISSN: 13891286. DOI: 10.1016/j.comnet.2020.107575. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128620312184 (visited on 04/20/2021).

[74] P. Sun, J. Lan, J. Li, Z. Guo, and Y. Hu, "Combining Deep Reinforcement Learning With Graph Neural Networks for Optimal VNF Placement," *IEEE Communications Letters*, vol. 25, no. 1, pp. 176–180, Jan. 2021, ISSN: 1089-7798, 1558-2558, 2373-7891. DOI: 10.1109/LCOMM.2020.3025298. [Online]. Available: https://ieeexplore.ieee.org/document/9201405/ (visited on 11/02/2021).

[75] Y. Xie *et al.*, "Virtualized Network Function Forwarding Graph Placing in sdn and nfv-Enabled iot Networks: A Graph Neural Network Assisted Deep Reinforcement Learning Method," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021, ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2021.3123460. [Online]. Available: https://ieeexplore.ieee.org/document/9590522/ (visited on 11/02/2021).

[76] A. Rkhami, T. A. Quang Pham, Y. Hadjadj-Aoul, A. Outtagarts, and G. Rubino, "On the Use of Graph Neural Networks for Virtual Network Embedding," in *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, Montreal, QC, Canada: IEEE, Oct. 2020, pp. 1–6, ISBN: 978-1-72815-628-6. DOI: 10.1109/ISNCC49221.2020.9297270. [Online]. Available: https://ieeexplore.ieee.org/document/9297270/ (visited on 11/02/2021).

[77] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic Virtual Network Embedding: A Deep Reinforcement Learning Approach With Graph Convolutional Networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, Jun. 2020, ISSN: 0733-8716, 1558-0008. DOI: 10.1109/JSAC.2020.2986662. [Online]. Available: https://ieeexplore.ieee.org/document/9060910/ (visited on 11/05/2021).

[78] *3GPP - The 3rd Generation Partnership Project, A Global Initiative*, https://www.3gpp.org/, accessed May 2021. [Online]. Available: https://www.3gpp.org/ (visited on 06/05/2021).

[79] *Auto Scaling Overview - Amazon Web Services (AWS)*, https://amzn.to/3fzX3UJ, accessed May 2021. [Online]. Available: https://aws.amazon.com/autoscaling/ (visited on 05/31/2021).

[80] *Auto Scaling Amazon EC2 - Amazon Web Services (AWS)*, https://amzn.to/3wKtYM7, accessed May 2021. [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html (visited on 05/31/2021).

[81] *Autoscale in Microsoft Azure - Azure Monitor*, https://bit.ly/3wLoMrv, accessed May 2021. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview (visited on 05/31/2021).

[82] *Autoscaling - Google Cloud*, https://bit.ly/3wJxG8J, accessed May 2021. [Online]. Available: https://cloud.google.com/compute/docs/autoscaler (visited on 05/31/2021).

[83] *Auto Scaling - Huawei Cloud*, https://bit.ly/34xww4f, accessed May 2021. [Online]. Available: https://support.huaweicloud.com/en-us/productdesc-as/en-us_topic_0042018383.html (visited on 05/31/2021).

[84] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," in *Advances in neural information processing systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/3a0772443a0739141292a5429b952fe6-Paper.pdf.

[85] X.-Z. Wu and Z.-H. Zhou, "A Unified View of Multi-Label Performance Measures," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, Aug. 2017, pp. 3780–3788. [Online]. Available: http://proceedings.mlr.press/v70/wu17a.html.

[86] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[87] T. Calders and S. Jaroszewicz, "Efficient AUC Optimization for Classification," en, in *Knowledge Discovery in Databases: PKDD 2007*, J. N. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenič, and A. Skowron, Eds., vol. 4702, ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 42–53, ISBN: 978-3-540-74975-2 978-3-540-74976-9. DOI: 10.1007/978-3-540-74976-9_8. [Online]. Available: http://link.springer.com/10.1007/978-3-540-74976-9_8 (visited on 05/13/2021).

[88] F. Charte, A. Rivera, M. J. del Jesus, and F. Herrera, "A First Approach to Deal with Imbalance in Multi-label Datasets," in *Hybrid Artificial Intelligent Systems*, D. Hutchison *et al.*, Eds., vol. 8073, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 150–160, ISBN: 978-3-642-40845-8 978-3-642-40846-5. DOI: 10.1007/978-3-642-40846-5_16. [Online]. Available: http://link.springer.com/10.1007/978-3-642-40846-5_16 (visited on 05/27/2021).

[89] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Addressing imbalance in multilabel classification: Measures and random resampling algorithms," en, *Neurocomputing*, vol. 163, pp. 3–16, Sep. 2015, ISSN: 09252312. DOI: 10.1016/j.neucom.2014.08.091. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231215004269 (visited on 05/26/2021).

[90] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," en, in *Proceedings of the 23rd international conference on Machine learning - ICML '06*, Pittsburgh, Pennsylvania: ACM Press, 2006, pp. 233–240, ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143874. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1143844.1143874 (visited on 05/02/2021).

[91] T. Saito and M. Rehmsmeier, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," en, *PLOS ONE*, vol. 10, no. 3, G. Brock, Ed., e0118432, Mar. 2015, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0118432. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0118432 (visited on 05/13/2021).

[92] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training deep neural networks on imbalanced data sets," in *2016 International Joint Conference on Neural Networks (IJCNN)*, Vancouver, BC, Canada: IEEE, Jul. 2016, pp. 4368–4374, ISBN: 978-1-5090-0620-5. DOI: 10.1109/IJCNN.2016.7727770. [Online]. Available: http://ieeexplore.ieee.org/document/7727770/ (visited on 05/23/2021).

[93] B. Liu and G. Tsoumakas, "Synthetic Oversampling of Multi-label Data Based on Local Label Distribution," en, in *Machine Learning and Knowledge Discovery in Databases*, U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, and C. Robardet, Eds., vol. 11907, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 180–193, ISBN: 978-3-030-46146-1 978-3-030-46147-8. DOI: 10.1007/978-3-030-46147-8_11. [Online].

Available: http://link.springer.com/10.1007/978-3-030-46147-8_11 (visited on 05/26/2021).

[94] J. Ma, H. Zhang, and T. W. S. Chow, "Multilabel Classification With Label-Specific Features and Classifiers: A Coarse- and Fine-Tuned Framework," *IEEE Transactions on Cybernetics*, vol. 51, no. 2, pp. 1028–1042, Feb. 2021, ISSN: 2168-2267, 2168-2275. DOI: 10.1109/TCYB.2019.2932439. [Online]. Available: https://ieeexplore.ieee.org/document/8809380/ (visited on 04/30/2021).

[95] Y. Zhong, B. Du, and C. Xu, "Learning to reweight examples in multi-label classification," en, *Neural Networks*, S0893608021001106, Apr. 2021, ISSN: 08936080. DOI: 10.1016/j.neunet.2021.03.022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0893608021001106 (visited on 05/01/2021).

[96] A. N. Tarekegn, M. Giacobini, and K. Michalak, "A review of methods for imbalanced multi-label classification," en, *Pattern Recognition*, vol. 118, p. 107 965, Oct. 2021, ISSN: 00313203. DOI: 10.1016/j.patcog.2021.107965. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0031320321001527 (visited on 05/26/2021).

[97] M.-L. Zhang, Y.-K. Li, H. Yang, and X.-Y. Liu, "Towards Class-Imbalance Aware Multi-Label Learning," *IEEE Transactions on Cybernetics*, pp. 1–13, 2020, ISSN: 2168-2267, 2168-2275. DOI: 10.1109/TCYB.2020.3027509. [Online]. Available: https://ieeexplore.ieee.org/document/9262911/ (visited on 05/26/2021).

[98] B. Liu and G. Tsoumakas, "Making Classifier Chains Resilient to Class Imbalance," in *Proceedings of The 10th Asian Conference on Machine Learning*, J. Zhu and I. Takeuchi, Eds., ser. Proceedings of Machine Learning Research, vol. 95, PMLR, Nov. 2018, pp. 280–295. [Online]. Available: http://proceedings.mlr.press/v95/liu18c.html.

[99] H. He and Y. Ma, Eds., *Imbalanced learning: foundations, algorithms, and applications*. Hoboken, New Jersey: John Wiley & Sons, Inc, 2013, ISBN: 978-1-118-64633-5 978-1-118-64620-5 978-1-118-64627-4.

[100] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[101] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from Imbalanced Data Sets*, en. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-98073-7 978-3-319-98074-4. DOI: 10.1007/978-3-319-98074-4. [Online]. Available: http://link.springer.com/10.1007/978-3-319-98074-4 (visited on 05/26/2021).

[102] Martín Abadi *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. [Online]. Available: https://www.tensorflow.org/.

[103] F. Chollet *et al.*, *Keras*. 2015. [Online]. Available: https://keras.io.

[104] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[105] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

[106] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier Chains for Multi-label Classification," in *Machine Learning and Knowledge Discovery in Databases*, W. Buntine, M. Grobelnik, D. Mladenić, and J. Shawe-Taylor, Eds., vol. 5782, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 254–269, ISBN: 978-3-642-04173-0 978-3-642-04174-7. DOI: 10.1007/978-3-642-04174-7_17. [Online]. Available: http://link.springer.com/10.1007/978-3-642-04174-7_17 (visited on 06/04/2021).

[107] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier Chains: A Review and Perspectives," *Journal of Artificial Intelligence Research*, vol. 70, pp. 683–718, Feb. 2021, ISSN: 1076-9757. DOI: 10.1613/jair.1.12376. [Online]. Available: https://www.jair.org/index.php/jair/article/view/12376 (visited on 06/04/2021).

[108] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," en, *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://direct.mit.edu/neco/article/9/8/1735-1780/6109 (visited on 07/31/2021).

[109] J. Kim, M. El-Khamy, and J. Lee, "Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition," *arXiv:1701.03360 [cs]*, Jun. 2017, arXiv: 1701.03360. [Online]. Available: http://arxiv.org/abs/1701.03360 (visited on 07/31/2021).

[110] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured Sequence Modeling with Graph Convolutional Recurrent Networks," *arXiv:1612.07659 [cs, stat]*, Dec. 2016, arXiv: 1612.07659. [Online]. Available: http://arxiv.org/abs/1612.07659 (visited on 03/16/2022).

[111] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," en, *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14236. [Online]. Available: http://www.nature.com/articles/nature14236 (visited on 02/04/2020).

[112] B. Rozemberczki *et al.*, "PyTorch geometric temporal: Spatiotemporal signal processing with neural machine learning models," in *Proceedings of the 30th ACM international conference on information and knowledge management*, 2021, pp. 4564–4573.

[113] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602 [cs]*, Dec. 2013, arXiv: 1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602 (visited on 03/25/2022).

Appendix A

Exploratory Data Analysis

Since the data we have worked with has a high dimensionality with over 27 million data points, this makes it difficult to visualize in a 2-dimensional space. Even when we label the data with the SLA and SLO definitions as elaborated upon in Chapter 3, that results on a multi-label format, i.e. each feature vector corresponding to a timestamp may be associated with more than one SLO label concurrently. This adds on to the complication of visualization of SLA/SLO clusters, since overlapping color coding and a dense consolidation of very high number of data points in a small 2D space reduce the efficacy of the visualization. To combat this, during the process of exploratory data analysis (EDA), we visualiaze the data two ways – using principal component analysis (PCA), and feature engineering using KPIs.

A.1 Principal Component Analysis (PCA)

PCA is a methodology involving the transformation of a high-dimensional data-set a new set of features called principal components (PCs), using orthogonal projection. The new set of features (principal components) typically present a lower-dimensional data-set, since a small subset of PCs possess the capability of capturing a significant proportion of the variance as present in the otherwise high-dimensional data-set with its original features. This is useful for visualising and processing high-dimensional data-sets, while still retaining as much of the variance in the data-set as possible. Figure presents an inverted scree plot that presents the variance explained by the PCAs when using standard scaler, and robust scaler respectively.

With standard scaler, the first 88 PCs are required to capture 99% of the variance in the original data-set. However, with robust scaler, the first 8 PCs equivalently capture the same information. If PCA is chosen from a dimensionality reduction perspective, the methodology used to standardize the data-set becomes an important variable. Standardization of a

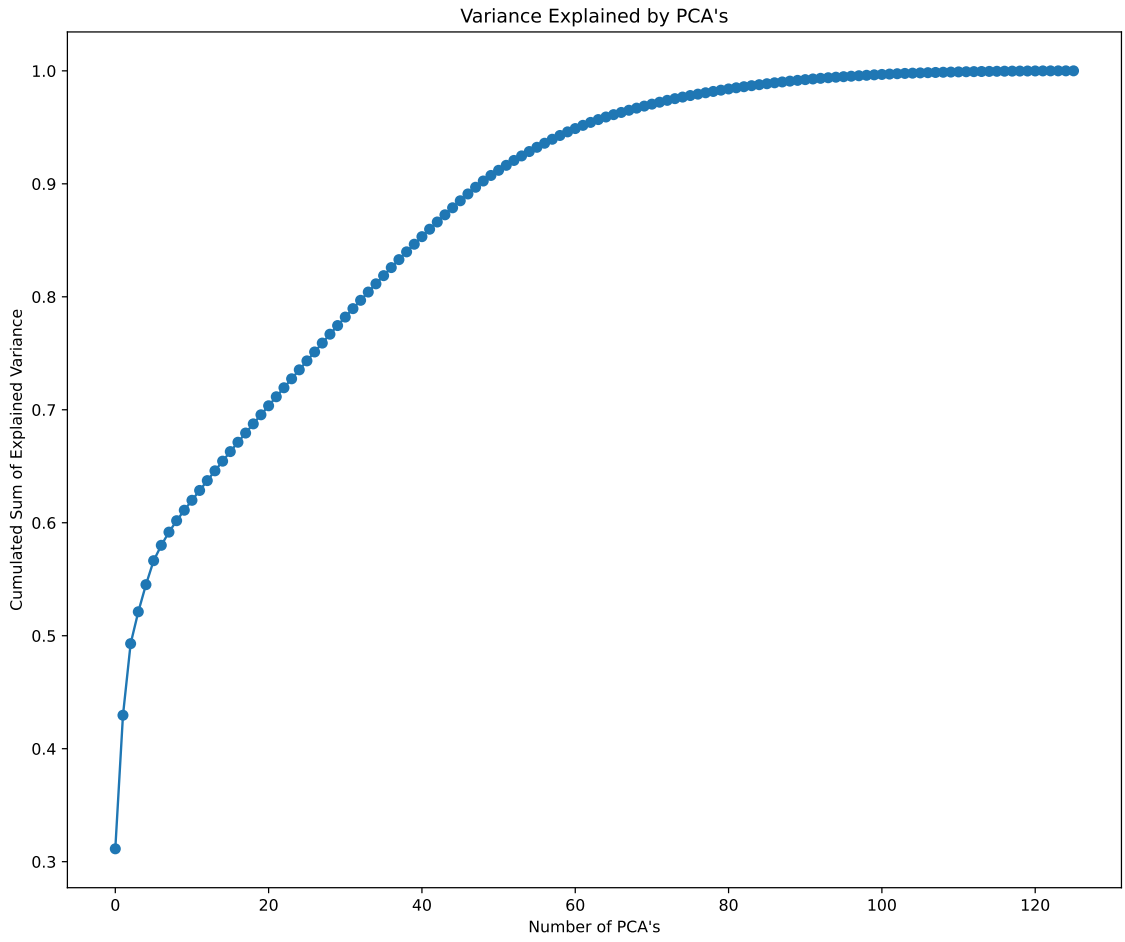


Fig. A.1 Inverted scree plot that coveys the cumulative sum of explained variance as captured by the various principal components when using standard scaler to standardize the data.

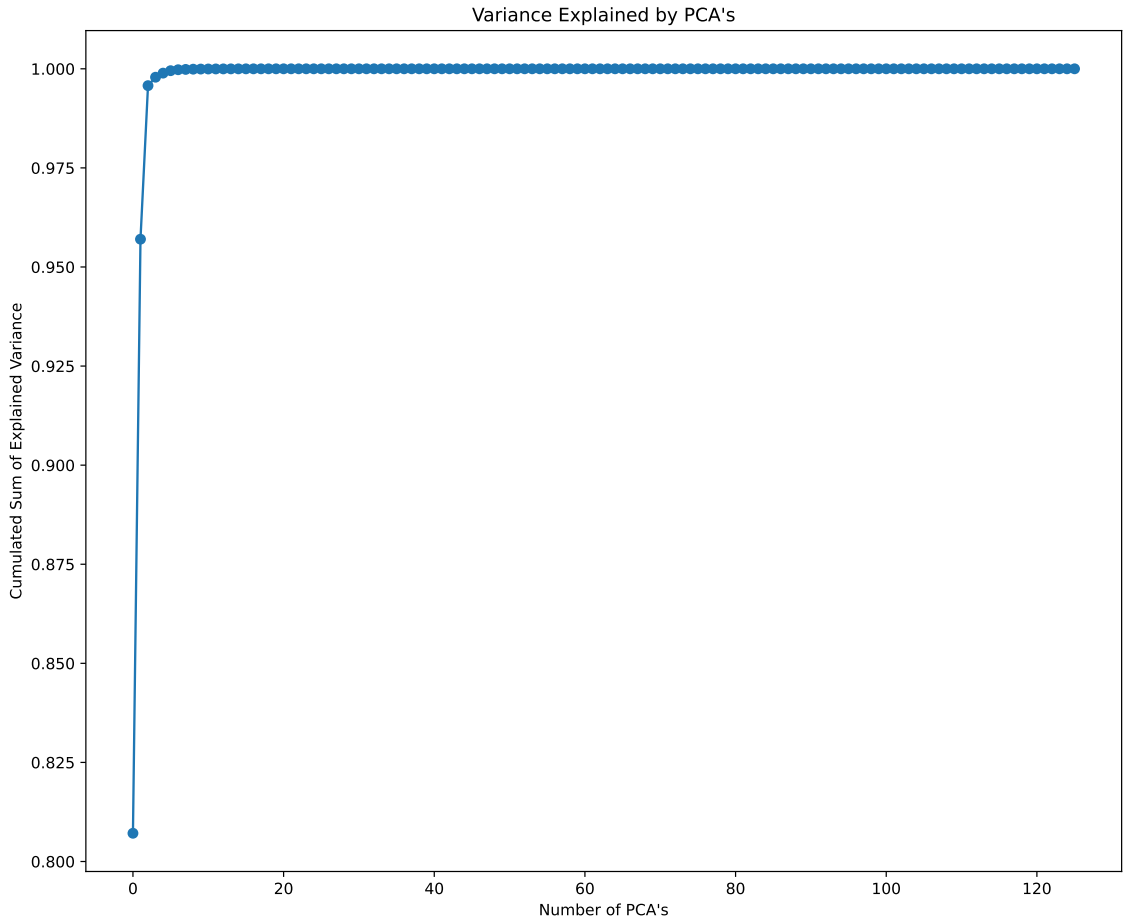


Fig. A.2 Inverted scree plot that coveys the cumulative sum of explained variance as captured by the various principal components when using robust scaler to standardize the data..

data-set is a common requirement for many machine learning estimators. Typically this is done by removing the mean and scaling to unit variance. However, outliers can often influence the sample mean/variance in a negative way. In such cases, the median and the interquartile range often give better results. Inherently, robust scaler removes the median and scales the data according to the interquartile quantile range, i.e. the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). This makes it somewhat resistant to outliers, which are very prevalent in our use-case data.

Figures A.3, A.5, A.6 present a 2D visualization of the SLA violation clusters using the first two PCs using various standardization techniques, and Figure A.4 presents a corresponding view of Figure A.3 when represented in 3D. Note that these representations still have limitations as listed above— the clusters are superimposed in 2D from highest prevailing SLO cluster to the lowest to ensure visualization, but there exist overlapping color codings (multi-label, i.e. the data-point represented as a violet dot may also be pink/grey, but the visualization here only shows it as violet due to the 2D overlap). A dense consolidation of very high number of data points in a small 2D space eliminates the possibility to use any other 2D visualizations that may have been more suitable for multi-label data.

To eliminate the loss of information during this stage of additional pre-processing via dimensionality reduction, and test the capabilities of the models on true features, our work as presented in this dissertation does not involve any dimensionality reduction.

A.2 Feature Engineering

Since the trend in each of the 21 features of the 6 Clearwater VNFCs is hard visually to interpret directly, we integrate general domain knowledge to feature engineer 7 KPIs that visually captures this information instead, in Figures A.7, A.8, A.9, A.10, A.11, A.12, and A.13. This is used to both understand the data, and the impact of the stress tests on the different VNFC node functions and categories that would be further captured in the SLA definitions we later formulate.

A.3 Correlation Analysis

To understand the correlation of features against others, we performed a visual analysis via correlation matrices that capture this information for the Clearwater VNFCs. This was one of the factors taken into account when drafting the feature-engineered KPIs as mentioned in §A.2. Figures A.14, A.15, A.16, A.17, A.18, and A.19 present the correlation matrices for each of the each of the 6 Clearwater VNFCs. Each square within the correlation matrix

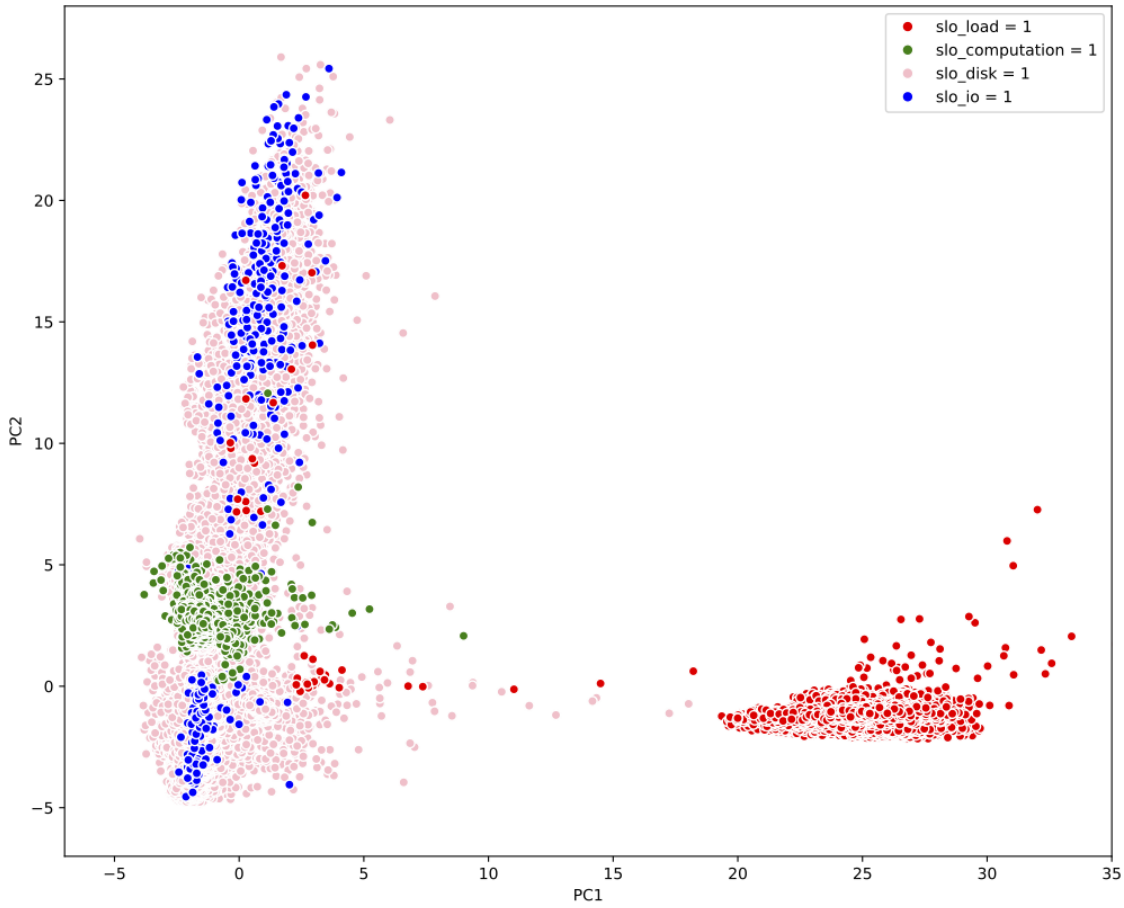


Fig. A.3 SLO violation clusters in data visualised in 2D— using PCA with a standard scaler.

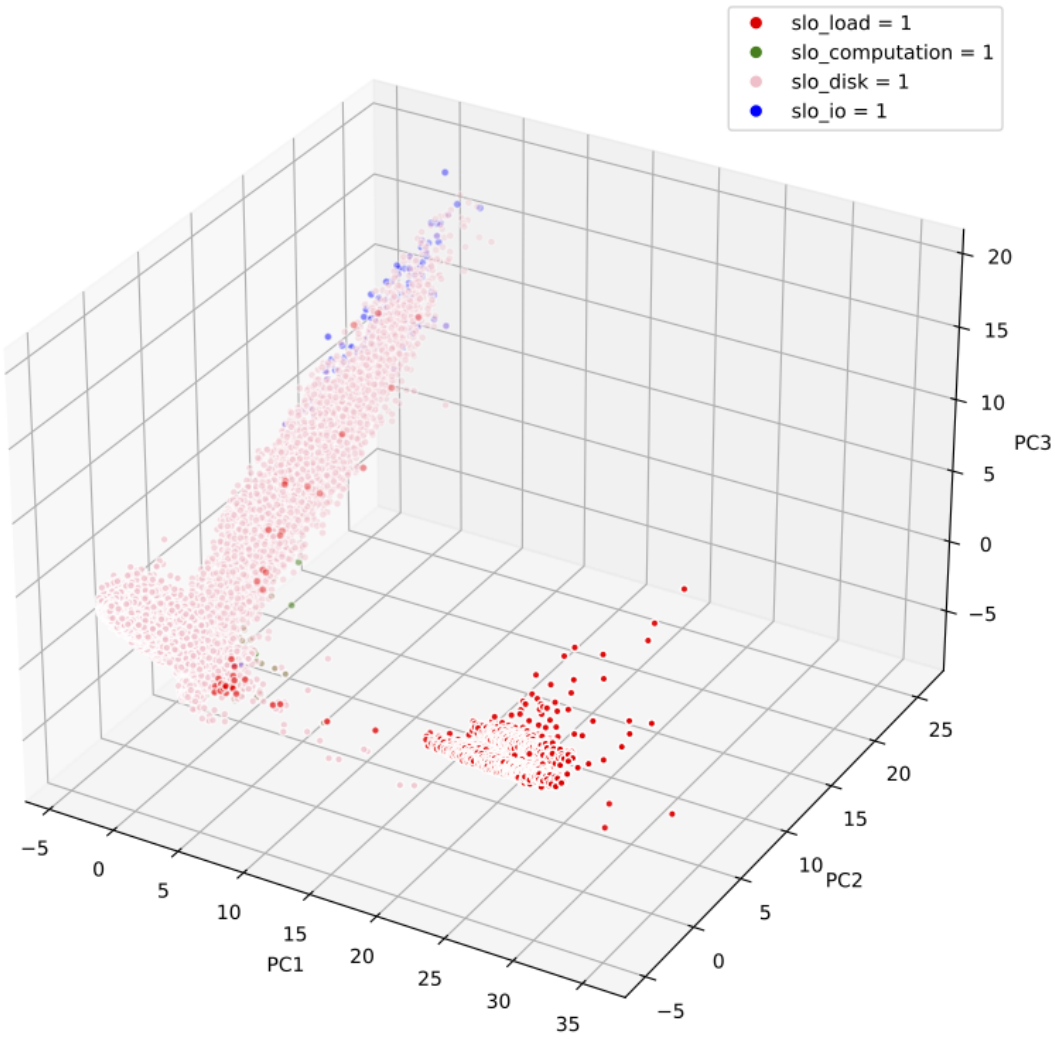


Fig. A.4 A view of the SLO violation clusters in data visualised in 3D— using PCA with a standard scaler.

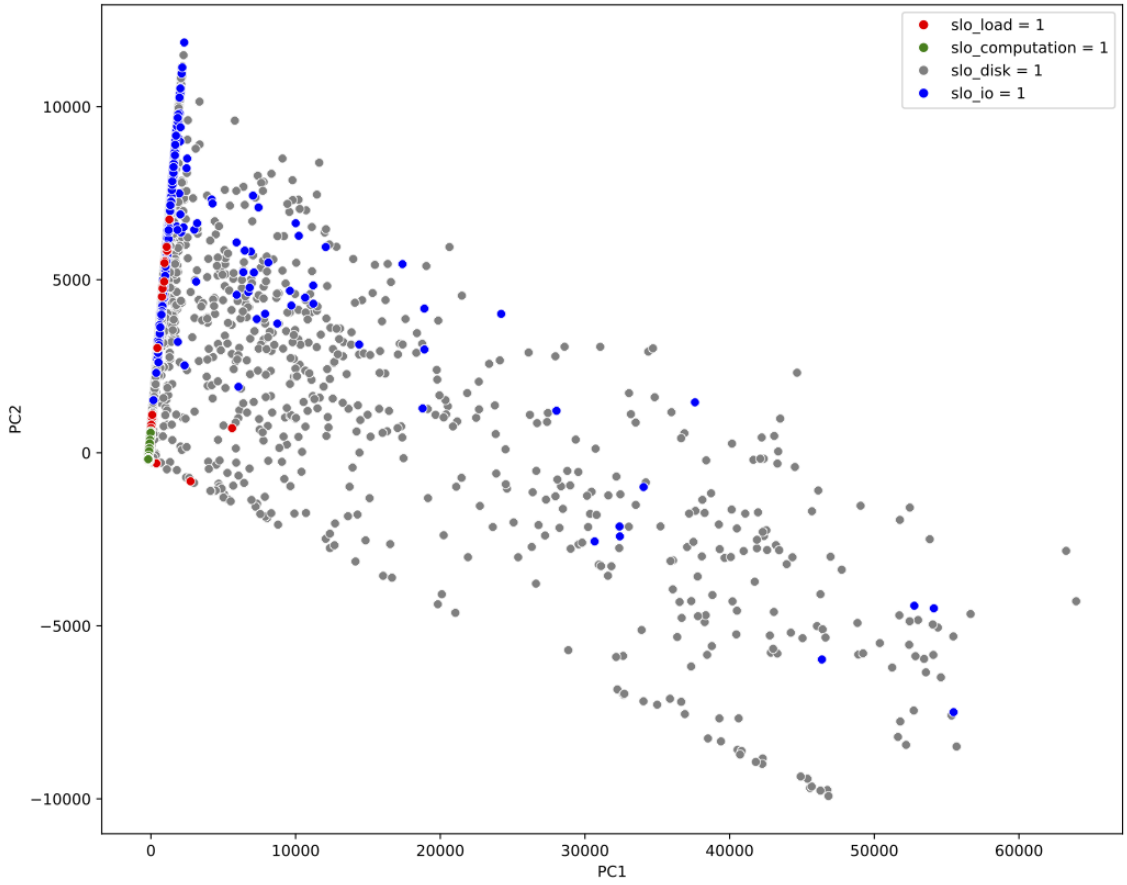


Fig. A.5 SLO violation clusters in data visualised in 2D— using PCA with a robust scaler.

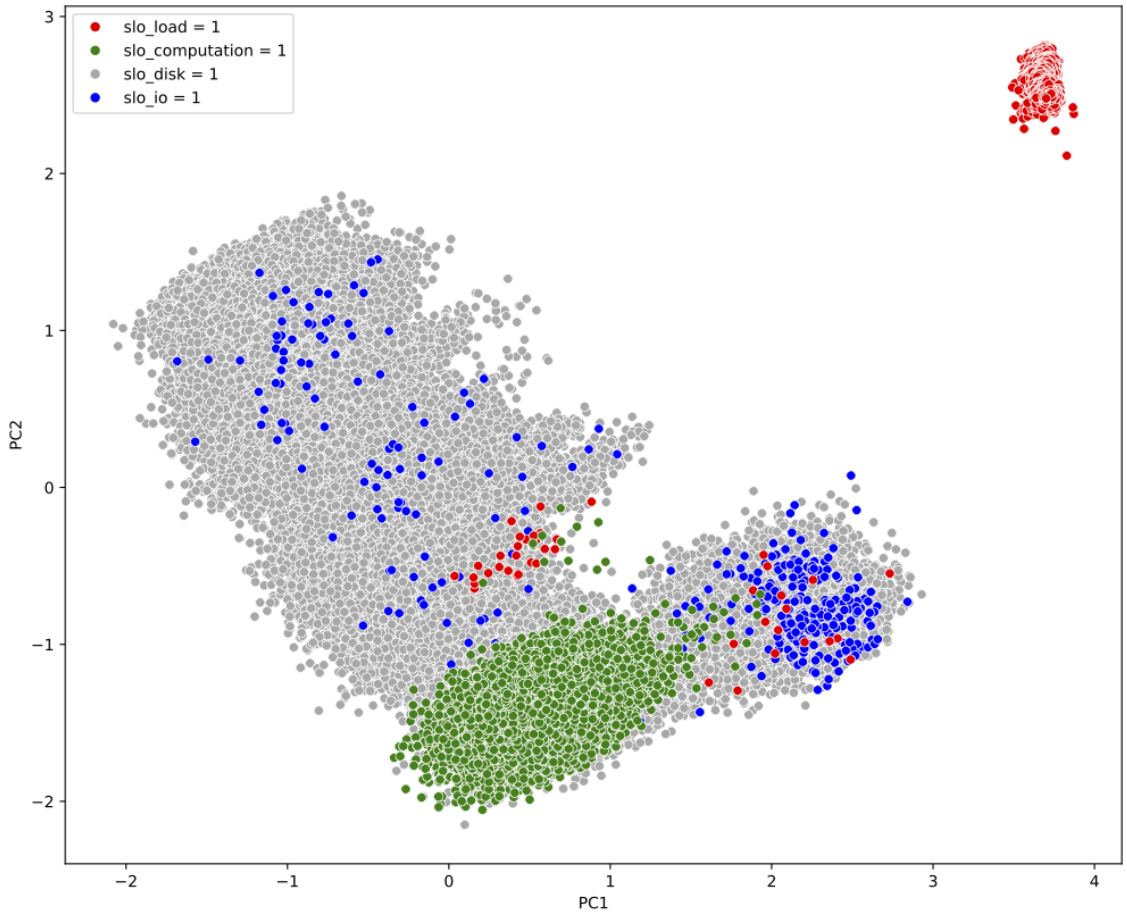


Fig. A.6 SLO violation clusters in data visualised in 2D— using PCA with a uniform quantile transformer.

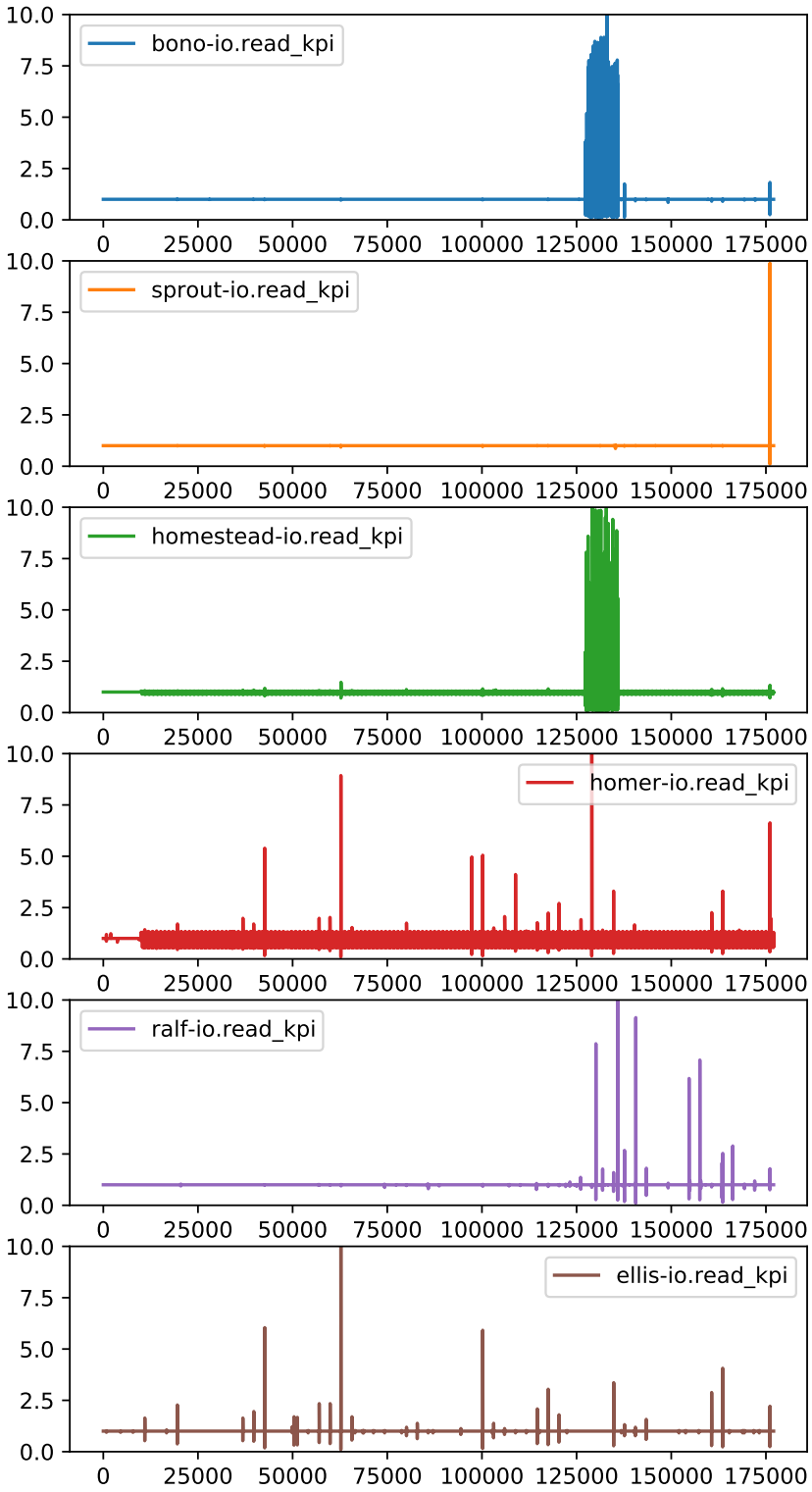


Fig. A.7 KPI – IO (Read). This is defined as *"Kbytes per sec read by an IO device / Number of read requests per sec to an IO device"*. y-axis represents the value of this KPI with a visual cap of 10, and x-axis represents discrete time-counts (rows within data-set).

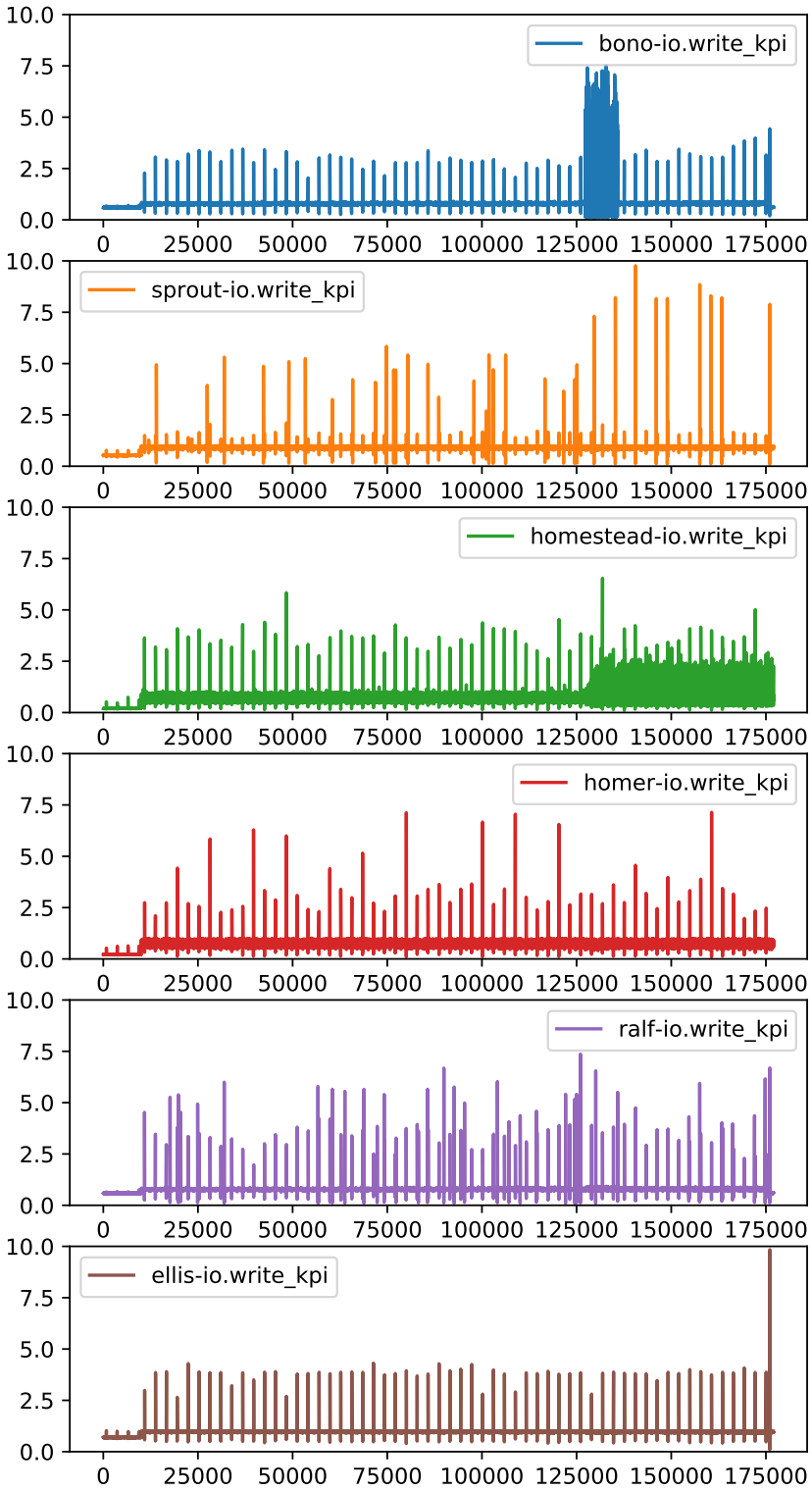


Fig. A.8 KPI – IO (Write). This is defined as *Kbytes per sec written by an IO device / Number of write requests per sec to an IO device*. y-axis represents the value of this KPI with a visual cap of 10, and x-axis represents discrete time-counts (rows within data-set).

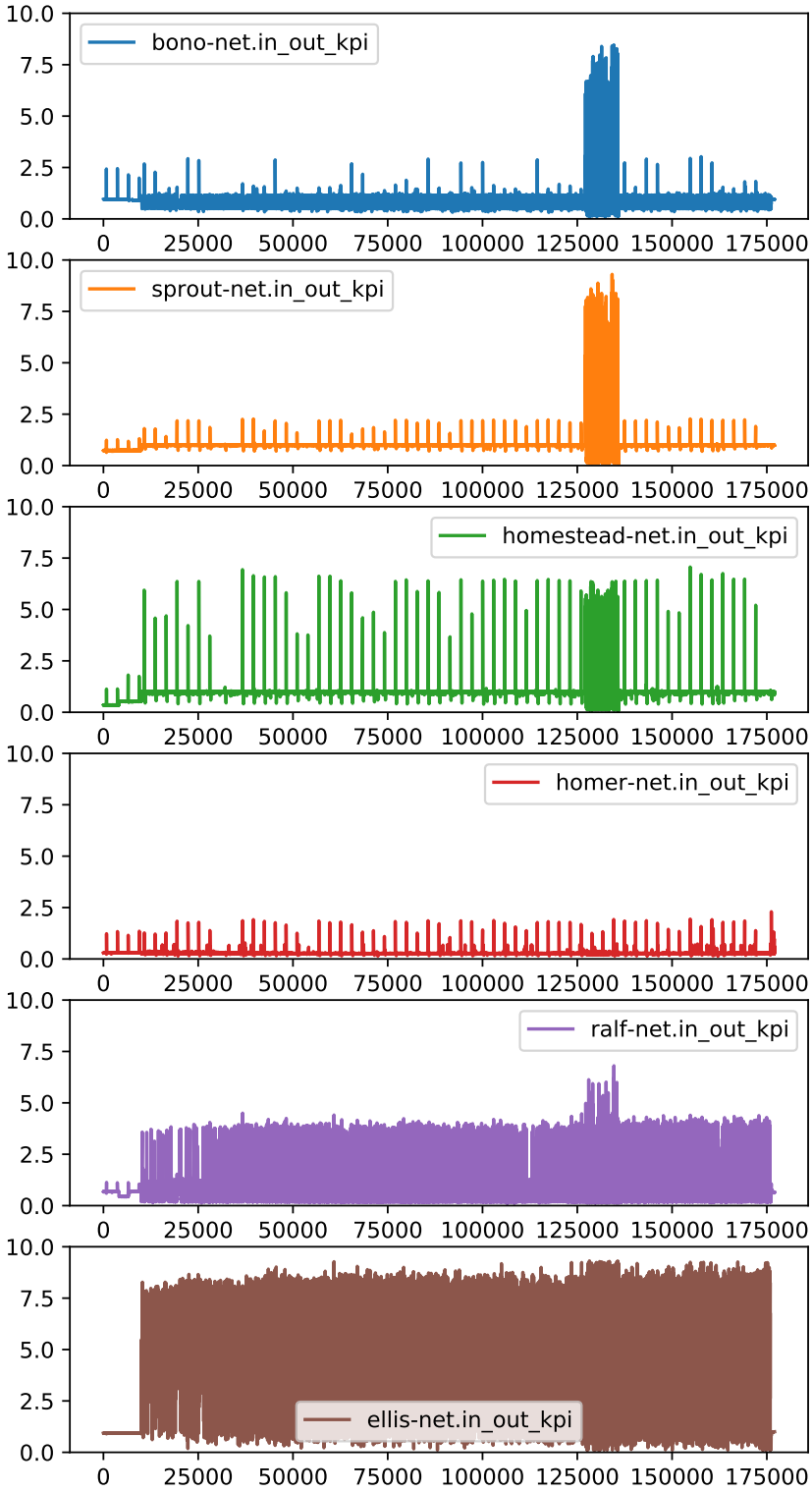


Fig. A.9 KPI – Network (In-Out). This is defined as *"Number of network bytes received per second / Number of network packets sent per second"*. y-axis represents the value of this KPI with a visual cap of 10, and x-axis represents discrete time-counts (rows within data-set).

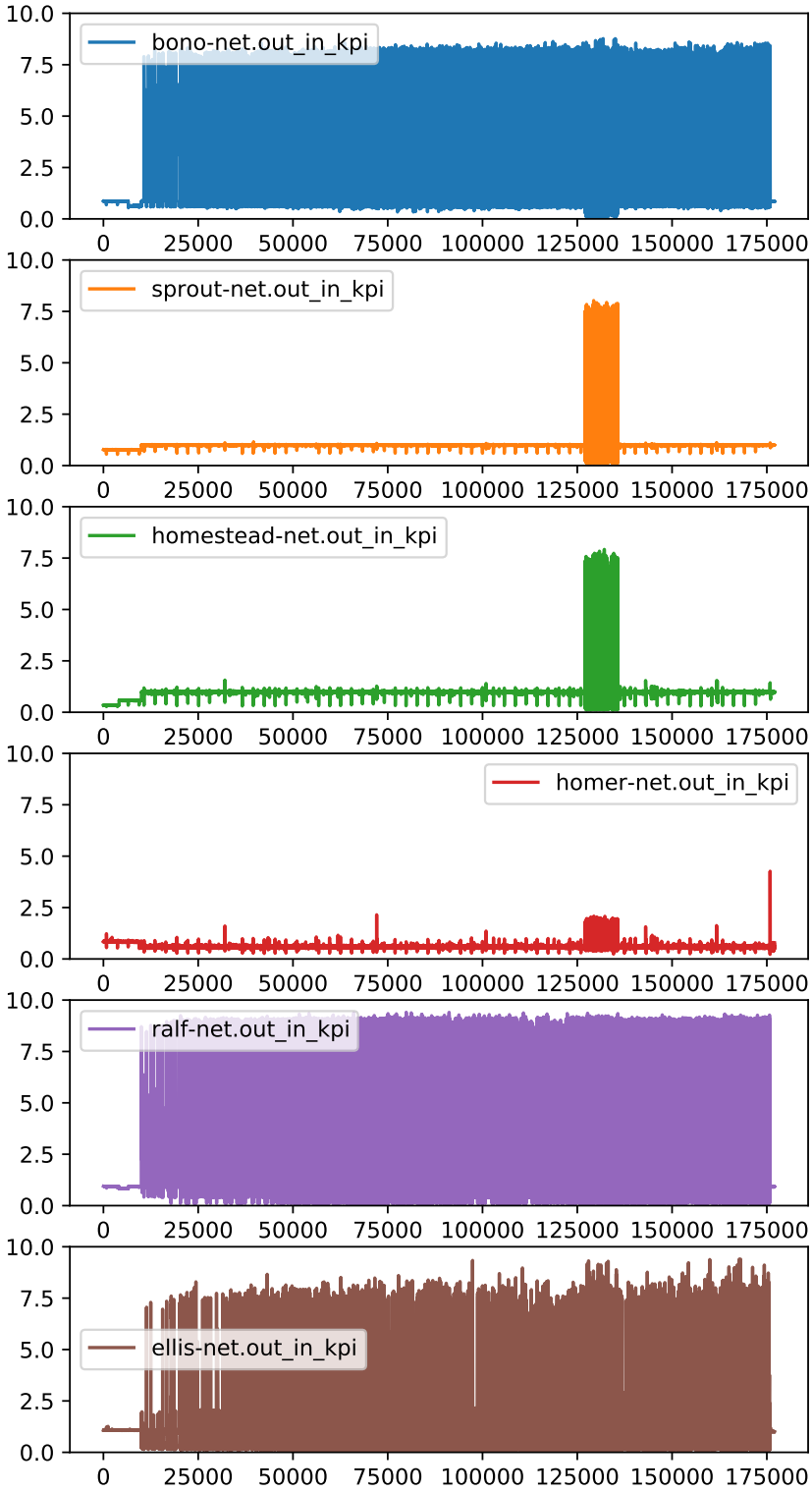


Fig. A.10 KPI – Network (Out-In). This is defined as *Number of network bytes sent per second / Number of network packets received per second.* y-axis represents the value of this KPI with a visual cap of 10, and x-axis represents discrete time-counts (rows within data-set).

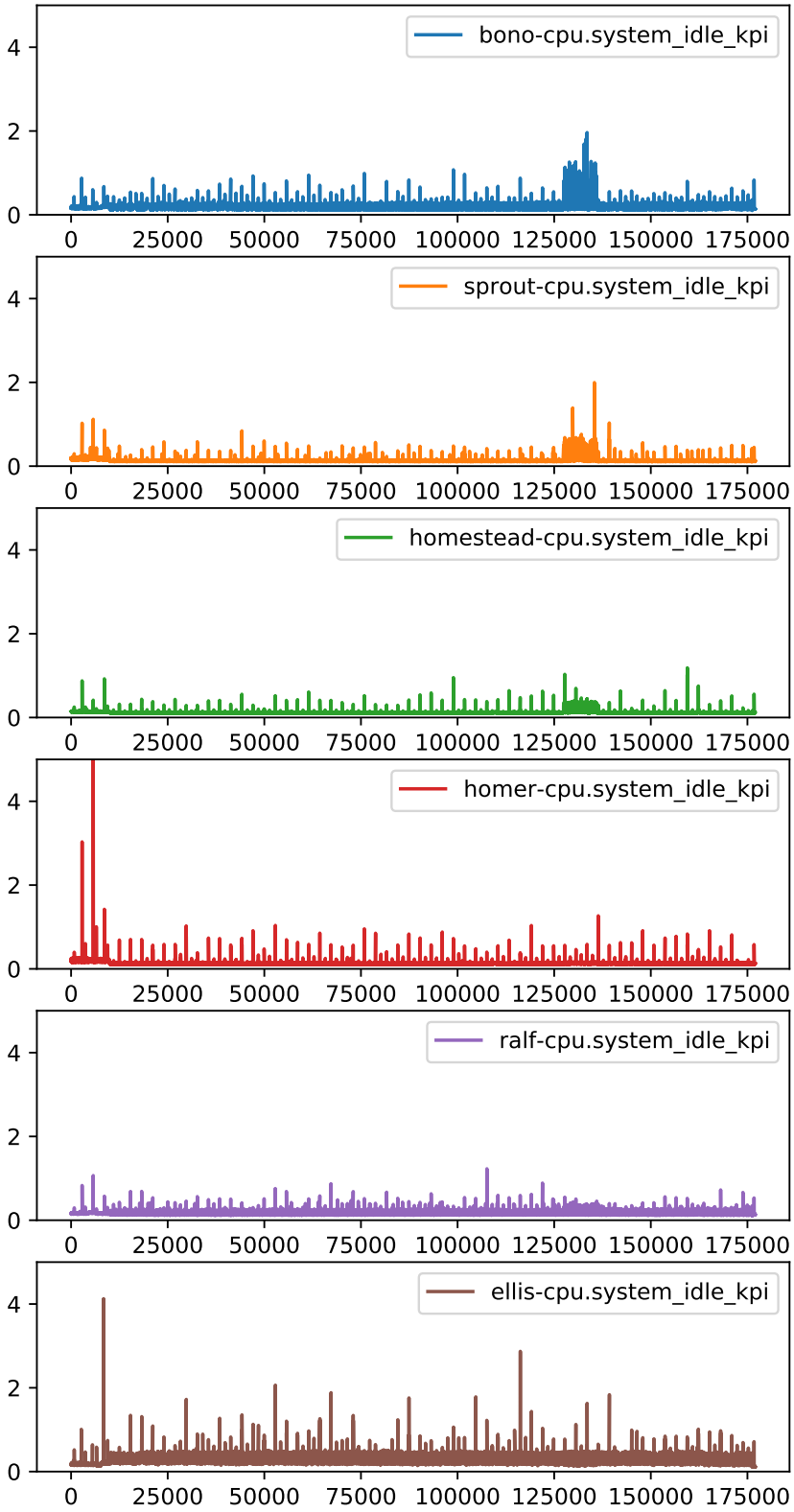


Fig. A.11 KPI – CPU. This is defined as *"% of time the CPU is used at the system level / % of time the CPU is idle when no IO requests are in progress"*. y-axis represents the value of this KPI with a visual cap of 10, and x-axis represents discrete time-counts (rows within data-set).

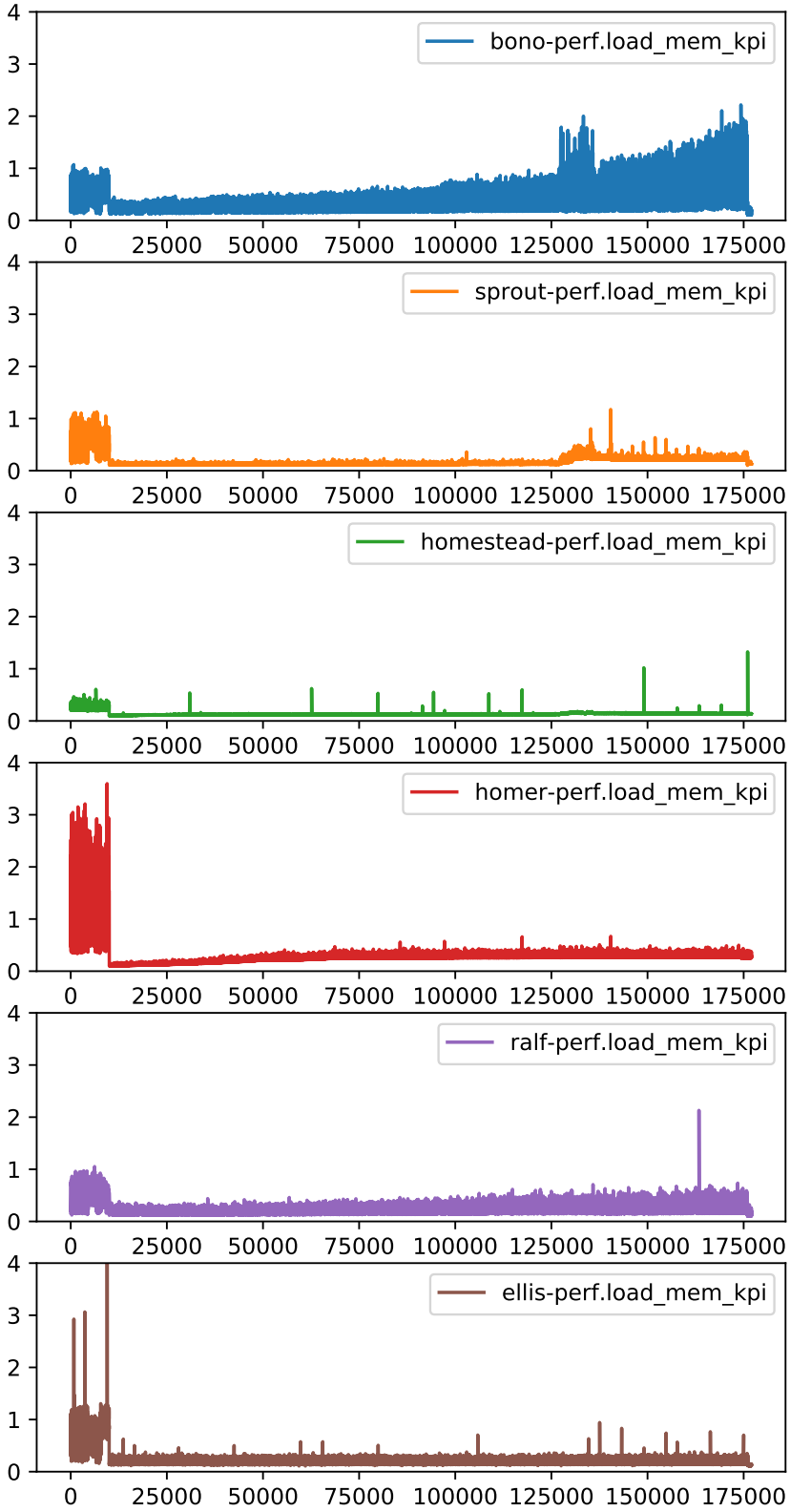


Fig. A.12 KPI – Performance (Load and Memory). This is defined as *"The normalized (by number of logical cores) average system load over a 1 minute period / Total Mbytes of usable memory"*. y-axis represents the value of this KPI with a visual cap of 10, and x-axis represents discrete time-counts (rows within data-set).

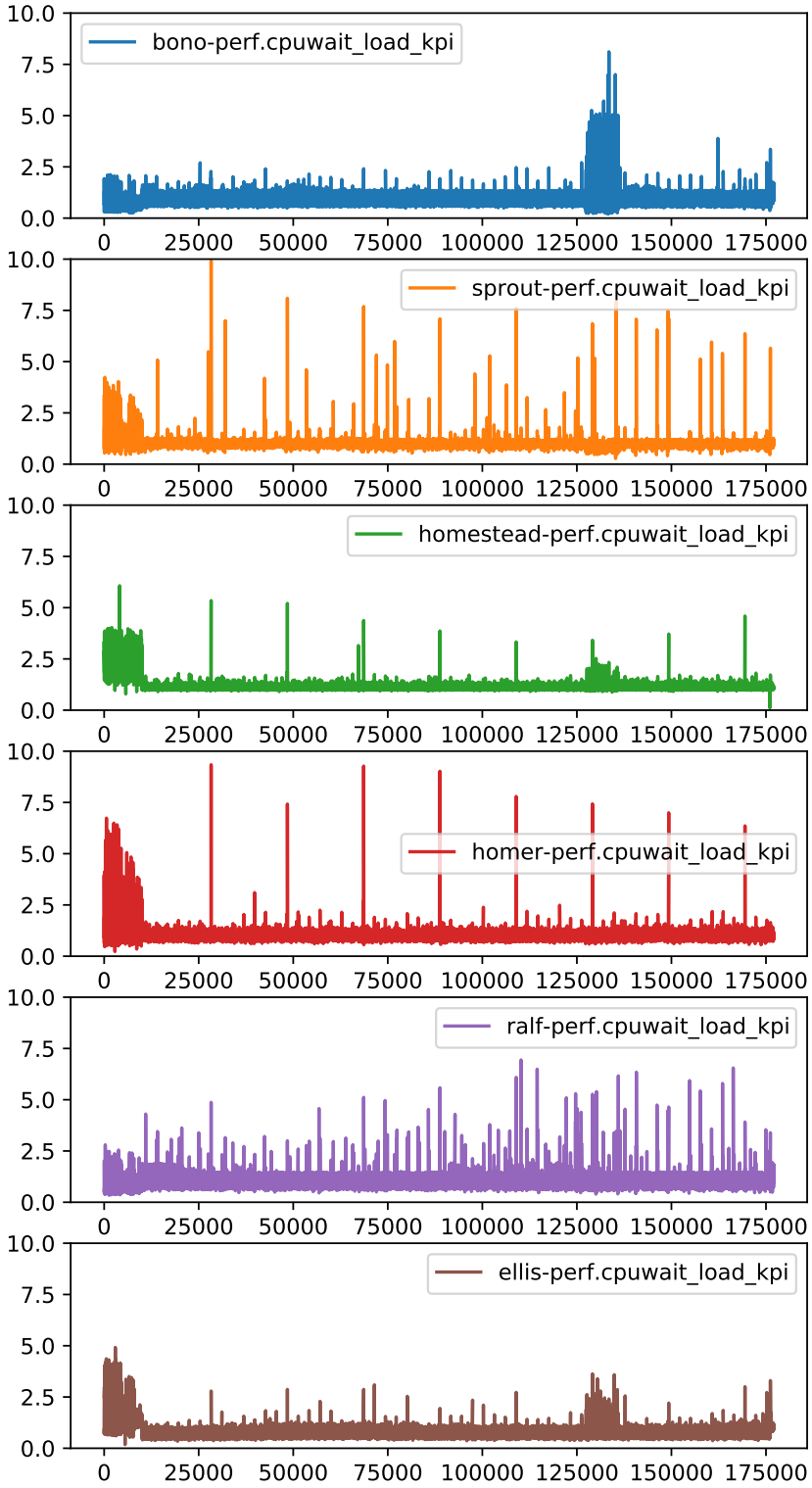


Fig. A.13 KPI – Performance (CPU Wait and Load). This is defined as *"% of time the CPU is idle AND there is at least one I/O request in progress / The normalized (by number of logical cores) average system load over a 1 minute period"*. y-axis represents the value of this KPI with a visual cap of 10, and x-axis represents discrete time-counts (rows within data-set).

represents a visual representation of the correlation coefficient r via a heat-map – deep red represents $r = +1$ (strong positive correlation), deep blue represents $r = -1$ (strong negative correlation), and white represents $r = 0$ (no correlation). While analyzing the correlation matrices, it is important to note that correlation does not imply causation.

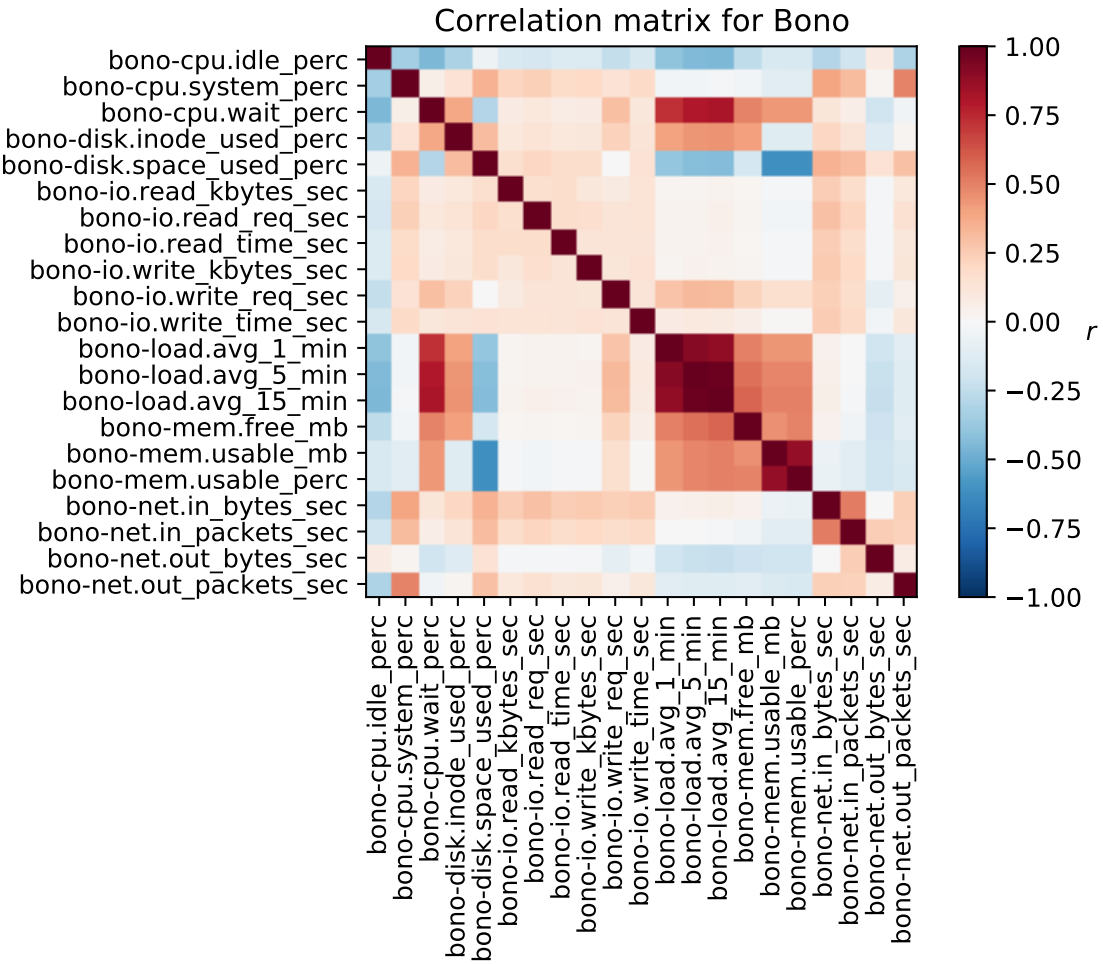


Fig. A.14 Correlation matrix for Bono.

A.4 Feature Scaling and Normalization for Forecasting Models

Standardization of a data-set is a common requirement for many machine learning estimators. Typically this is done by removing the mean and scaling to unit variance. However, the neural network based forecasting algorithms all perform best when the input features are normalized in the range of $[0, 1]$. As mentioned in chapters 4 and 5, this pre-processing is done via a Quantile Transformer, which is a non-linear transformation method to trans-

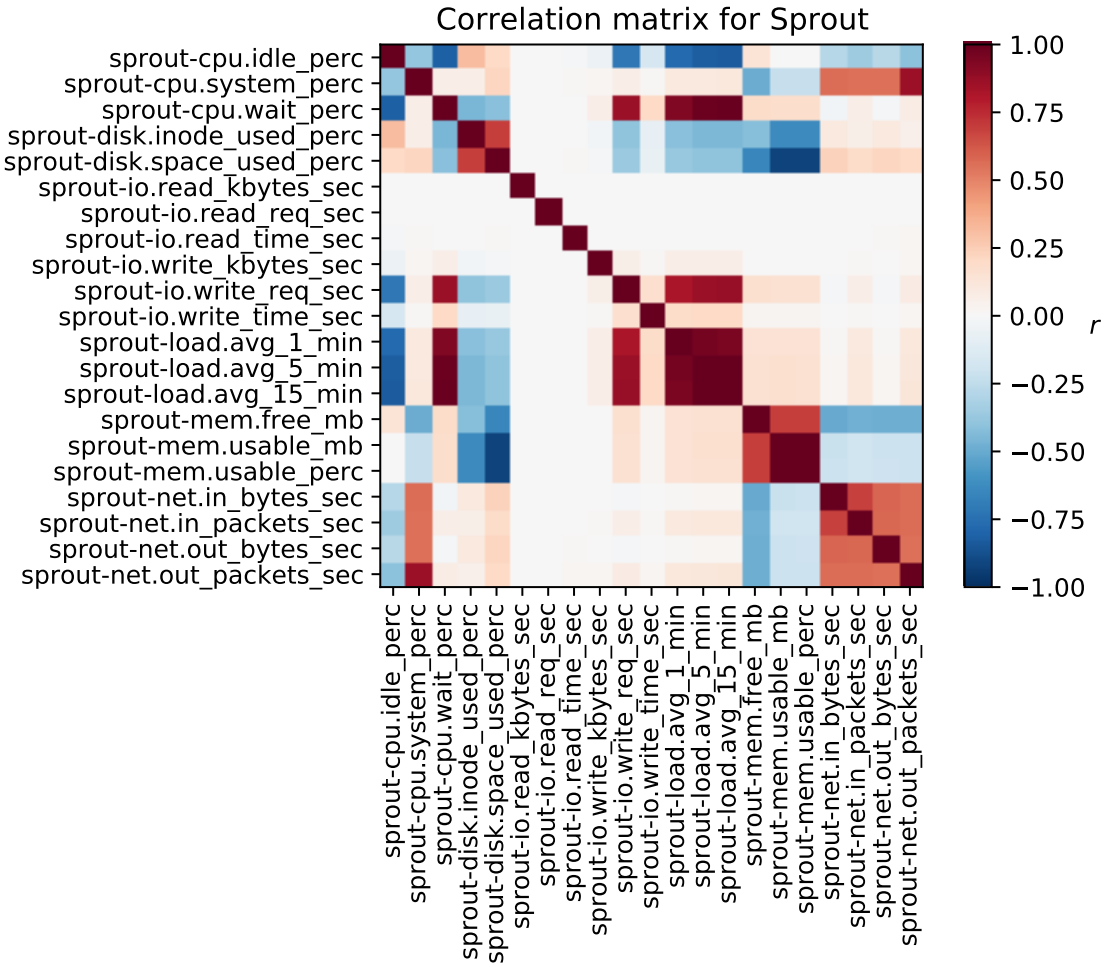


Fig. A.15 Correlation matrix for Sprout.

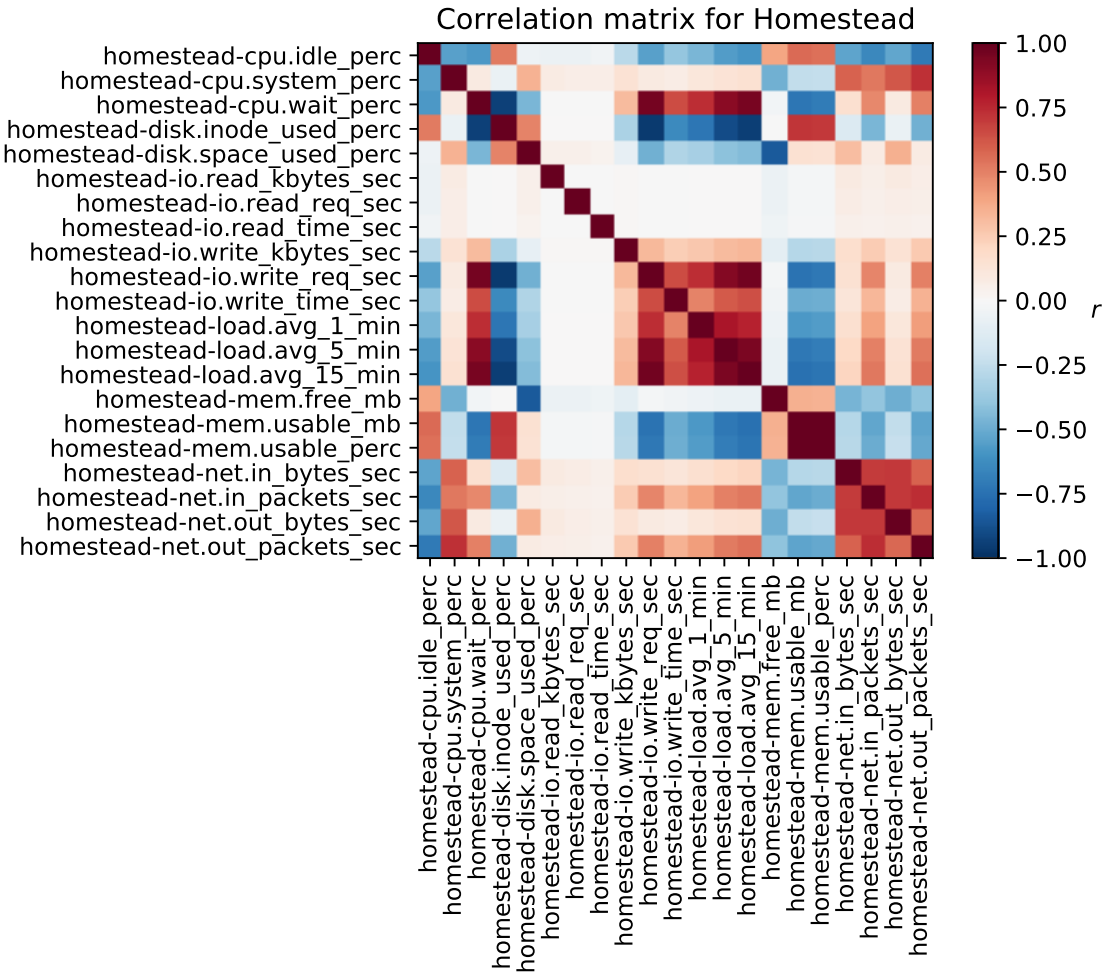


Fig. A.16 Correlation matrix for Homestead.

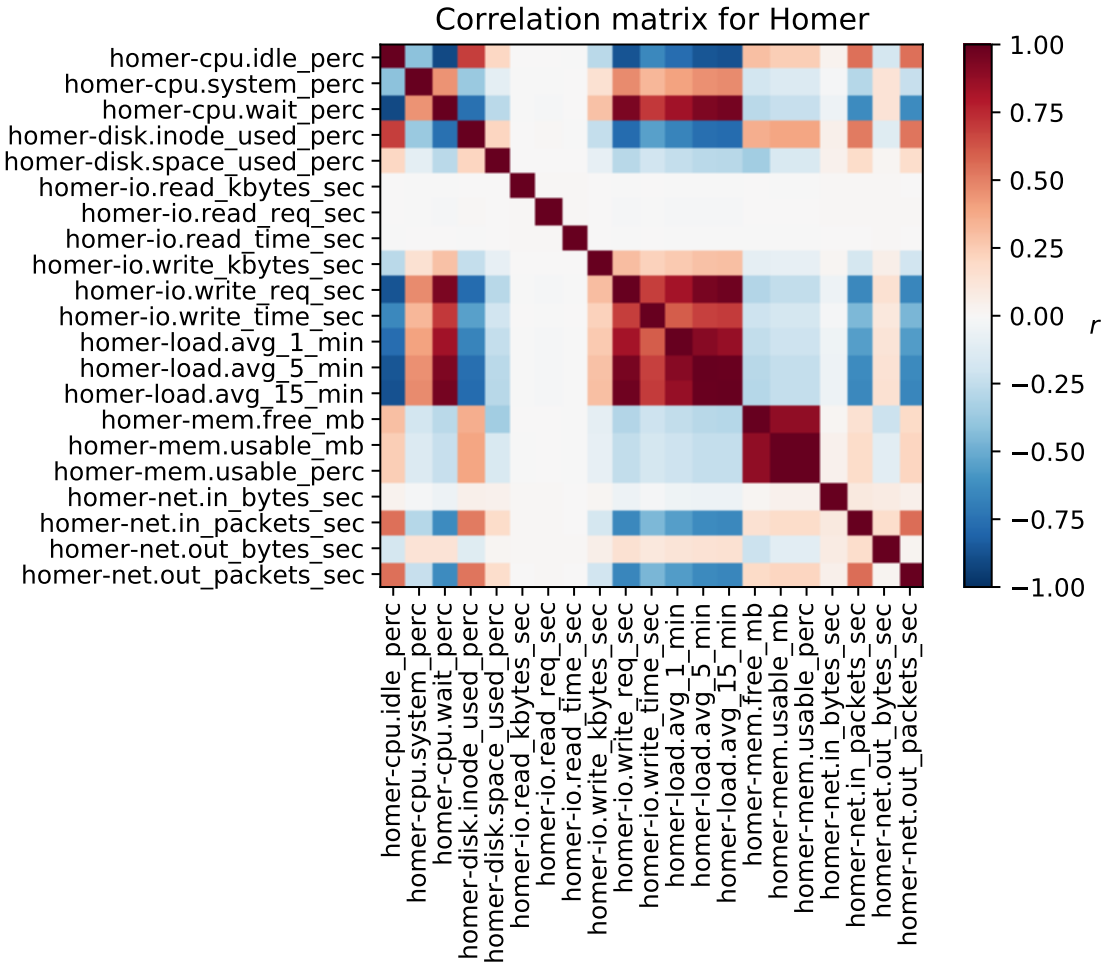


Fig. A.17 Correlation matrix for Homer.

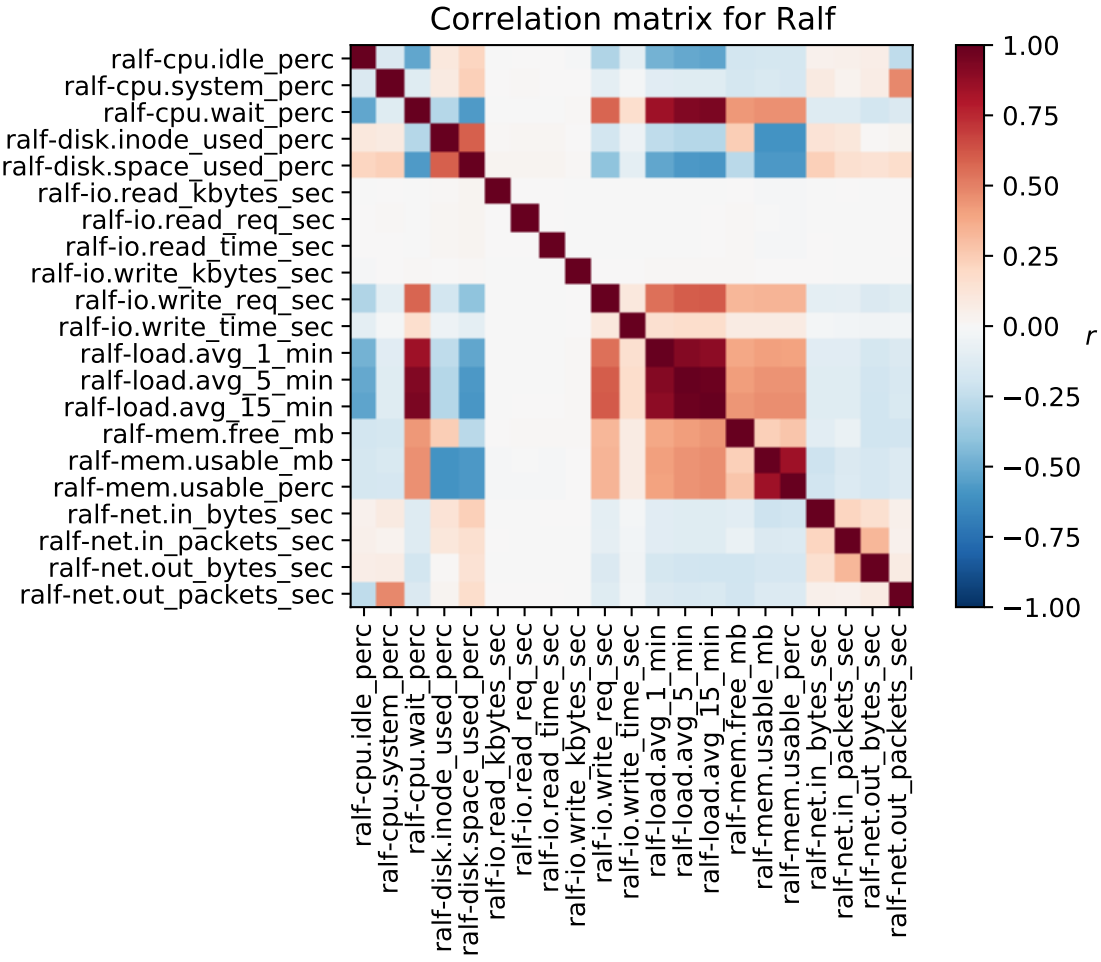


Fig. A.18 Correlation matrix for Ralf.

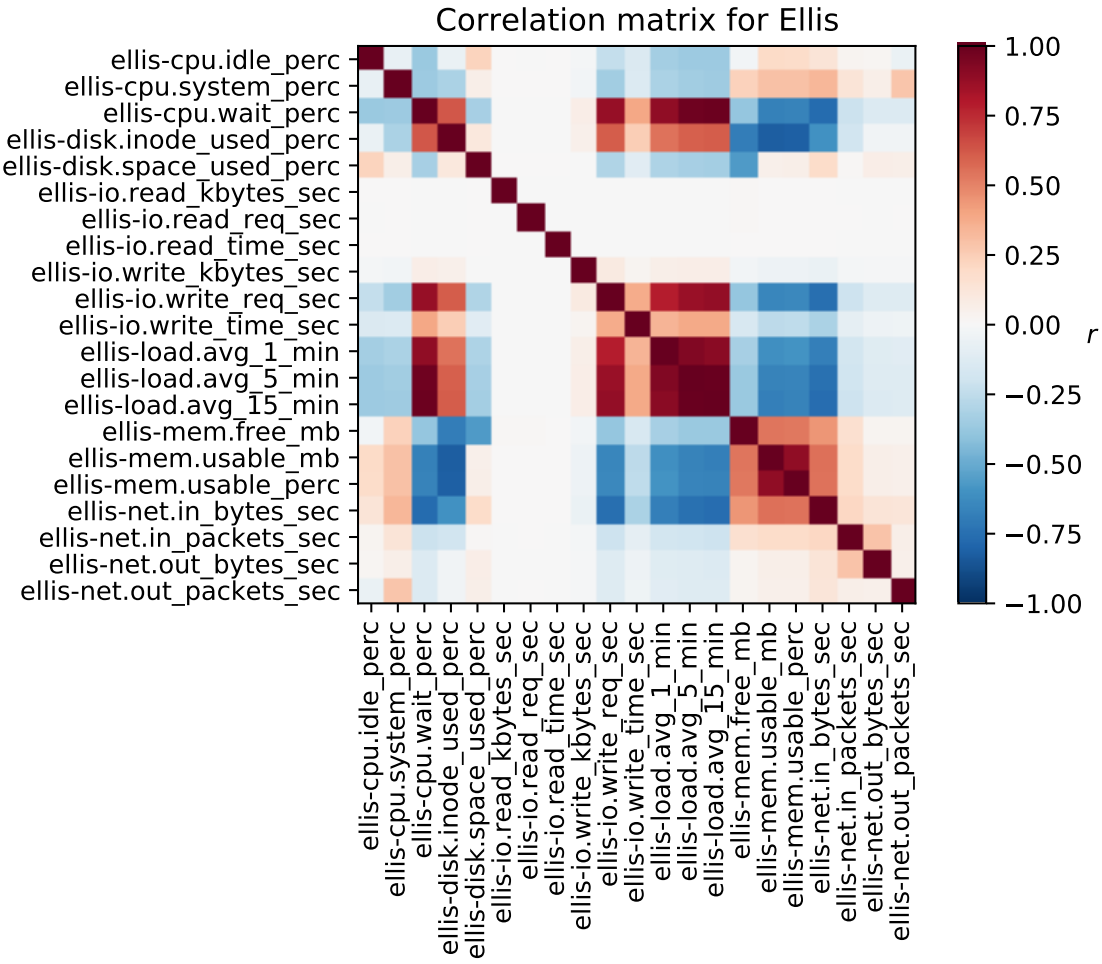


Fig. A.19 Correlation matrix for Ellis.

form the very skewed nature of this data-set and map it to a uniform distribution in $[0, 1]$. The resulting feature distribution is as visualized in the sample violin plots in Figures A.20, A.21, and A.22. The long tails in the distributions represent the outliers due to the incorporated stress tests, which need to be incorporated within the data during prepossessing steps to capture abnormal trends that contribute to SLA violations.

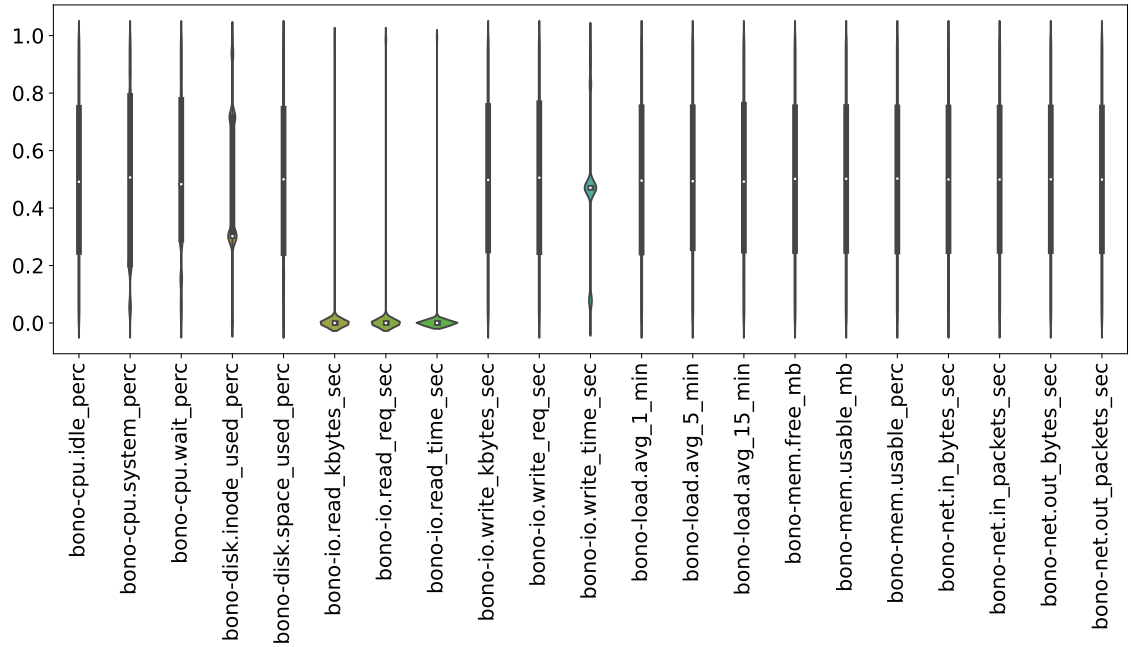


Fig. A.20 Violin plots representing the distribution of Bono after pre-processing via a quantile transformer.

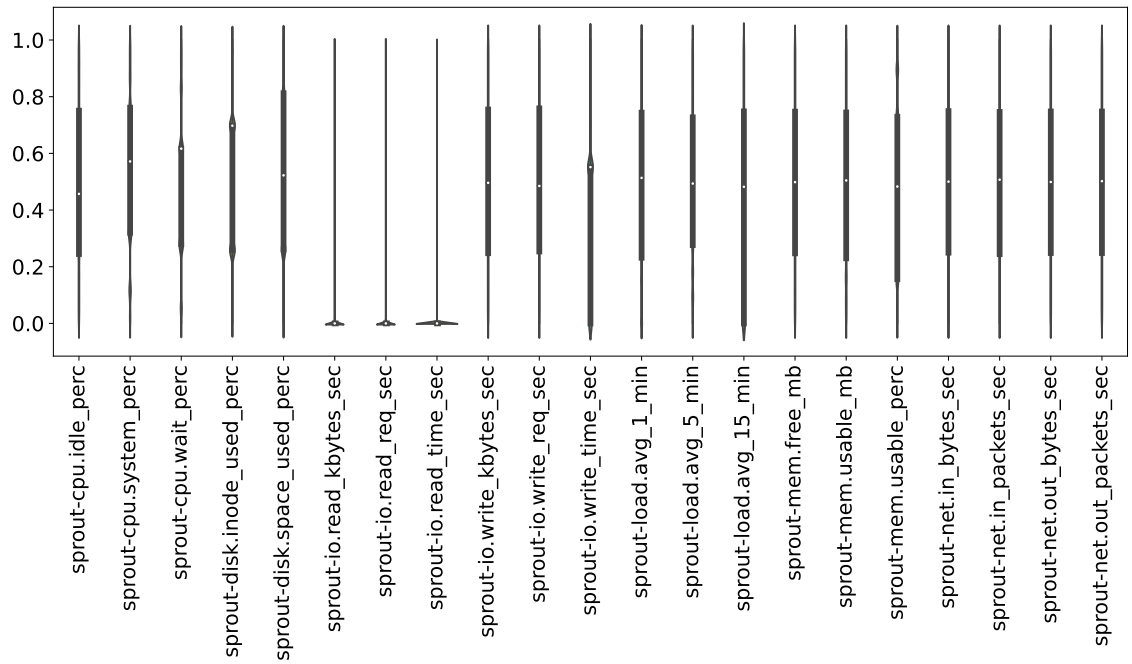


Fig. A.21 Violin plots representing the distribution of Sprout after pre-processing via a quantile transformer.

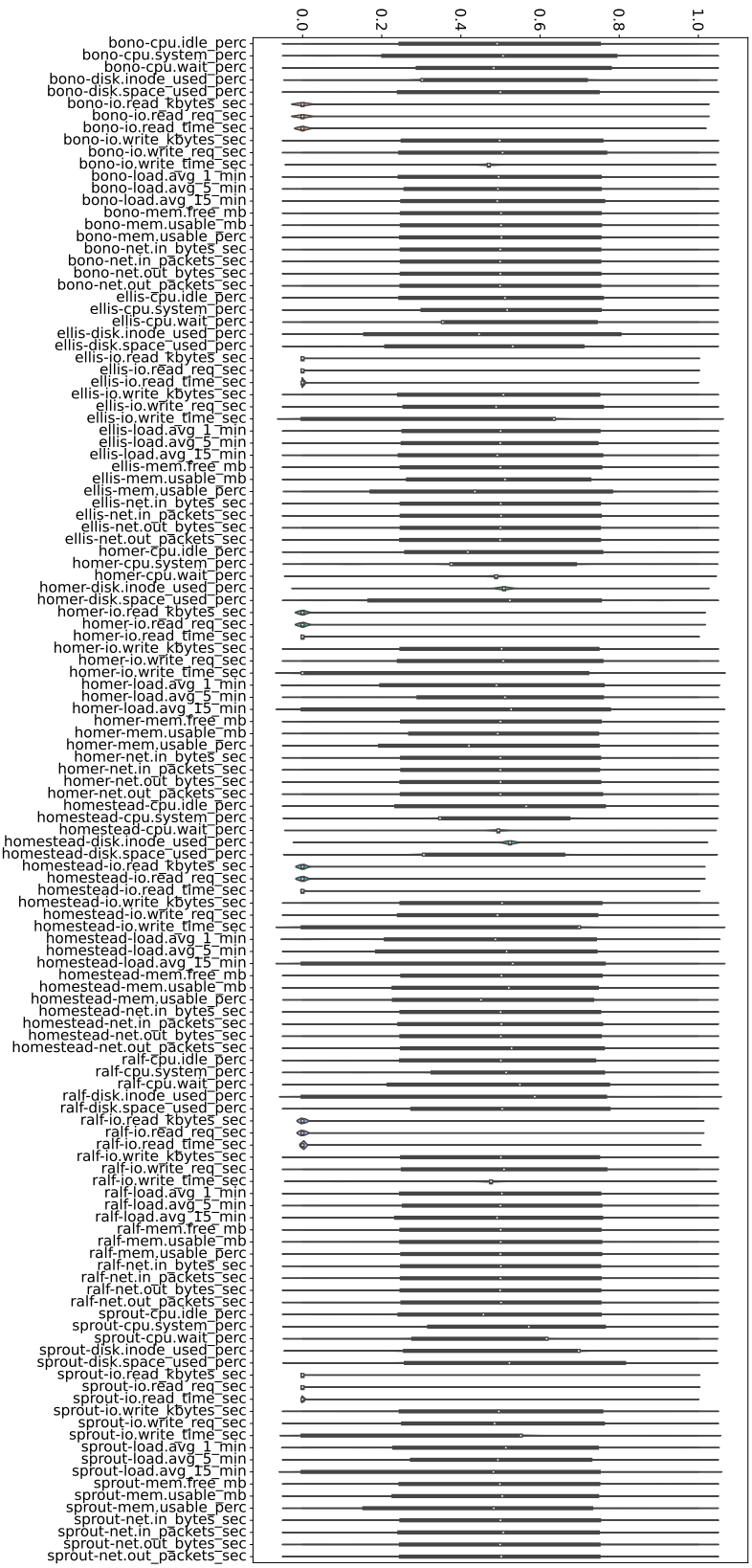


Fig. A.22 Violin plots representing the distribution of the data-set after pre-processing via a quantile transformer.

Appendix B

Deep Reinforcement Learning for Topology-Aware VNF Resource Prediction in NFV Environments

This publication **[Publication 4 – IEEE NFV-SDN 2019 [7]]** was disseminated as an early proof-of-concept regarding the integration of reinforcement learning and traditional graph neural networks on the use-case. This was released to seek early peer-review and subject-area feedback while contemplating simulation environments, before narrowing down the scope to the real-world dataset.

While this has contributions to how the following work shaped-up in scope, it does not directly feature in the material as presented in the key chapters of the dissertation.

Conference Title	2019 5th IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)
Article Type	Regular Paper (Fast-Track)
Complete Author List	Nikita Jalodia, Shagufta Henna and Alan Davy
Status	Published in the conference proceedings
Scope	Early proof of concept
Dissertation Reference	[Publication 4 – IEEE NFV-SDN 2019 [7]]

Deep Reinforcement Learning for Topology-Aware VNF Resource Prediction in NFV Environments

Nikita Jalodia*,†, Shagufta Henna*,†, Alan Davy*,†
*Telecommunications Software and Systems Group, Waterford Institute of Technology, Waterford, Ireland
†CONNECT- Centre for Future Networks and Communications, Dublin, Ireland
{njalodia, shenna, adavy}@tssg.org

***Abstract*—Network Function Virtualisation (NFV) has emerged as a key paradigm in network softwarisation, enabling virtualisation in future generation networks. Once deployed, the Virtual Network Functions (VNFs) in an NFV application's Service Function Chain (SFC) experience dynamic fluctuations in network traffic and requests, which necessitates dynamic scaling of resource instances. Dynamic resource management is a critical challenge in virtualised environments, specifically while balancing the trade-off between efficiency and reliability. Since provisioning of virtual infrastructures is time-consuming, this negates the Quality of Service (QoS) requirements and reliability criterion in latency-critical applications such as autonomous driving. This calls for predictive scaling decisions to balance the provisioning time sink, with a methodology that preserves the topological dependencies between the nodes in an SFC for effective resource forecasting. To address this, we propose the model for an Asynchronous Deep Reinforcement Learning (DRL) enhanced Graph Neural Networks (GNN) for topology-aware VNF resource prediction in dynamic NFV environments.**

***Index Terms*—NFV, Graph Neural Networks, Deep Reinforcement Learning, Asynchronous Deep Q-Learning, Dynamic Resource Prediction, Future Generation Networks, Topology Awareness, Prediction, Machine Learning, Deep Learning**

I. Introduction

Telecommunications has been one of the oldest industries serving humankind, with communication being the backbone of our existence. Telecom operators have the most complex operations and business support systems (OSS/BSS), with a vast array of dedicated infrastructure resources provisioned overtime to keep up with the demand and supply. These dedicated hardware resources are not only plagued with maintenance and operational overheads, but are also quite limited in flexibility. The upcoming application architectures across supported verticals on the Internet come with an extreme set of requirements, including ultra-low latency and high reliability. In the current overhaul of existing systems, Network Function Virtualization (NFV) [1] has emerged as a revolutionary paradigm for future networks and communications starting 5G and beyond [2]. By decoupling Network Functions (NF) and dedicated Network Appliances (NA), NFV allows NFs to evolve independently from hardware, thus leading towards reduced Capital and Operational Expenditure (CAPEX/OPEX).

Server placement, function placement, and dynamic resource management are the three key problem areas when looking at resource allocation in NFV [3]. While the first two have been widely studied [1], there are still challenges with dynamic resource management in NFV [3]. Efficiency and reliability are the key trade-offs while allocating resources to Virtual Network Functions (VNFs) and VNF Components (VNFCs) in an NFV application's Service Function Chain (SFC), an example of which is shown in Fig. 1. Dynamic resource management in NFV refers to dynamic scaling of deployed VNFs in the SFC subject to real-time network traffic. Specific to horizontal scaling, the time taken to boot and provision a Virtual Machine (VM) by the most popular Virtual Infrastructure Managers (VIMs) is in the order of tens of seconds. As studied in [4], this gets particularly worse when considering scalability, as the provisioning time worsens exponentially while spinning up multiple VM instances in this

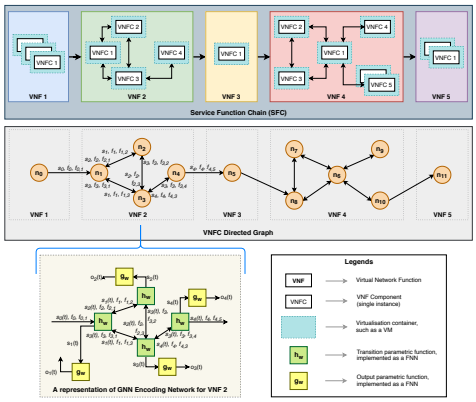


Fig. 1. A representation of an NFV application's SFC, with directed links between VNFs forming the VNF-FG (forwarding graph). Each VNF is composed of smaller scalable software components called VNFCs, which provide a sub-set of that VNFs functionality. VNFCs within a VNF are linked to each other by directed and undirected links in a vendor-specific topology, and work together to deliver the required functionality of the VNF. This is followed by a directed graph representation of an SFC topology, at the granularity of VNFCs. The GNN diffusion process for exchange of state and features is highlighted for a subset of the graph. Also represented is the encoding network formed, as per the GNN modeling mechanism described in section II.

This work has been funded by Science Foundation Ireland (SFI) and the European Regional Development Fund under Grant Number 13/RC/2077.

978-1-7281-4545-7/19/$31.00 ©2019 IEEE

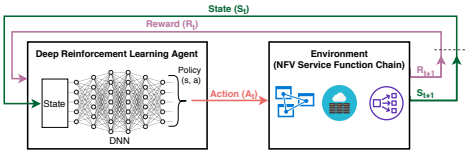


Fig. 2. A high level architecture of a DRL agent's interaction with the environment, which in this case is the NFV SFC. For DRL algorithms such as Deep Q-Learning, deep neural networks (DNN) act as non-linear function approximators to learn policy and state value representations. Actions here represent scaling decisions, that affect the state of the nodes in the SFC.

scenario. Given that 5G would drive applications across key latency critical verticals such as autonomous driving, smart grid, etc., reliability is the key. However, over-provisioning the systems to ensure reliability at all times not only adds to the CAPEX and OPEX, but is also inefficient, unsustainable and related to unnecessary carbon emissions and critical resources fuelling data-centres.

Given that the network traffic and resource requirements ebb and flow in this era of digital connectivity, the amount of resources allocated to each VNF in a latency critical SFC cannot be based on traditional threshold-based scaling due to provisioning delay. There is a recognized need [5] for an automated dynamic scaling solution for allocating resources to VNFs whose load may vary over time. This requires scaling decisions to be predicted ahead of time, so that the required resources are up and running when traffic reaches this point in the SFC. Existing work [6] in the domain uses Graph Neural Networks (GNN) [7] to model the topological dependencies between the VNFCs in the VNF Forwarding Graph (VNF-FG) derived from the SFC of an open source IMS (IP Multimedia Subsystem) application. While this achieved a good performance over traditional scaling methods and reduced the call drops by scaling the required VNFCs ahead of time, there are still flaws. The accuracy of the system drops significantly when moving from training to new and test data, which implies a low generalisation accuracy. This is because the the system uses the classic backpropagation through time algorithm to train the GNN model, which is inefficient when using large SFC [6]. It is also heavily dependent on the training for the static input network, and would require complete retraining with the slightest change in the input network [7]. The approach also wouldn't respond well to anomalous dynamic network and traffic fluctuations that aren't covered in the training data.

When it comes to dynamic application scenarios, Reinforcement Learning (RL) is an effective machine learning methodology. As abstractly shown in Fig. 2, a RL agent directly interacts with the environment to form a policy for decision-making based on a reward mechanism, that is customizable to achieve the desired outcomes. However, given a complex interconnected topology with multiple components like in an SFC, traditional RL is unscalable due to the high dimensionality of outcomes. Deep Reinforcement Learning (DRL) however, uses Neural Networks (NNs) as function approximators to deal with a large range of outcomes and their impact over time, and is much better adept to deal with a complex environment with a big range of outcomes. Fig. 2 shows the high-level model of DRL when interacting with an NFV SFC.

However, when looking at an SFC at the granularity of the VNFCs as shown in Fig. 1, there is a clear effect of topological dependencies that DRL wouldn't be able to model by itself in a complex environment. Fig. 1 demonstrates a clear notion of a cascading domino effect – a single point of failure in the form of an overloaded VNFC with more traffic requests than it can address would directly affect the following nodes in the VNF-FG. This has a range of implications on the service provider bound by Quality of Service (QoS) and Service-Level Agreements (SLA) depending on the type of applications.

Thus, to address the above, we propose the model for an Asynchronous DRL enhanced GNN for topology-aware VNF resource prediction in dynamic NFV environments. Multiple DRL agents with asynchronous input methodology on learning and policy fit well into the different Points of Presence (PoPs) of VNFs in the SFC topology represented by GNN, and further help reduce learning and training overheads as compared to classical DRL.

The paper has been further structured as follows: §II describes the Graph Neural Network model for modeling topological dependencies, §III elaborates on Asynchronous Deep Reinforcement Learning for learning and adaptation of policy for dynamic resource prediction, §IV presents the proposed system architecture and links GNN and DRL with our use case, and §V presents the conclusion and future work.

II. Graph Neural Networks

The GNN model [7] is a supervised neural network model for graph and node focused applications, with a methodology based on constrained information diffusion and relaxation mechanisms. It extends Recurrent Neural Networks (RNNs) and Markov Chains to directly deal with generic graph structured information without losing topological dependencies. Unlike traditional machine learning methodologies, the GNN model preserves the topological dependencies by encoding the topological graph relationships during the pre-processing phase.

A. Mathematical Model

A graph G is represented as a pair (N, E), where N is the set of nodes, and E is the set of edges. Each node $n \in N$ has a set of features $f_n \in \mathbb{R}^{D_N}$, which in our case would be VNFC memory m_n, CPU c_n, processing delay d_n, etc. n^* represents

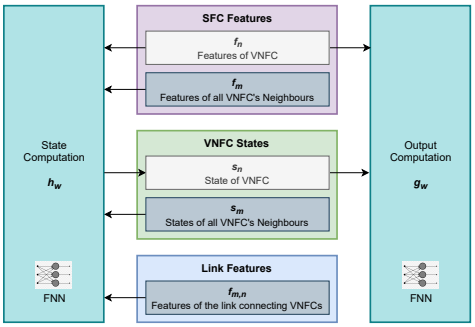


Fig. 3. A representation of the GNN workflow and diffusion mechanism to exchange state and features within all nodes in the neighbourhood, until equilibrium.

the nodes directly connected to n, which is referred as its neighbourhood. $f_{m,n} \in \mathbb{R}^{D_L}$ represents the link characteristics such as link delay $l_{m,n}$, bandwidth b_{nm} between two nodes n and m, where $m \in n^*$. These are then used to determine the state s_n for each node as shown in equation 1, which is then further used to calculate the output o_n as shown in equation 2.

At its core, the GNN is driven by two parametric functions h_w (transition function) and g_w (output function) that express the dependence of the state of each node on the state of its neighbourhood, and the dependence of the node output on its state, respectively.

$$s_n = \sum_{m \in n^*} h_w\big(f_n, f_m, f_{m,n}, s_m\big), \quad \forall n \tag{1}$$

$$o_n = g_w(s_n, f_n), \quad \forall n \tag{2}$$

Banach's fixed-point theorem [8] guarantees [7] the existence and uniqueness of a solution to equation 1 subject to the transition function being a contraction map with respect to s, and suggests a classic iterative scheme for computing the state, as defined in equation 3 below:

$$s_n(i+1) = \sum_{m \in n^*} h_w\big(f_n, f_m, f_{m,n}, s_m(i)\big), \quad \forall n \tag{3}$$

$s(i)$ here denotes the i^{th} iteration of s, and each iteration represents a discrete time step t. This makes the current state of a node dependent on the previous state of its neighbours, and this dynamical system converges exponentially fast to the solution of equation 2 for any initial value $s(0)$. The GNN model classically uses backpropagation through time methodology as its learning algorithm [7].

Fig. 3 describes the workflow of the GNN modeling, and indicates how the parameters interconnect in the diffusion mechanism.

B. Encoding Network

This computation as described in the equations above can be interpreted as the representation of a network consisting of units, which compute h_w and g_w. This is the encoding network. Each VNFC in the SFC as shown in Fig. 1 is thus modelled via two parametric functions h_w and g_w, each of which is implemented by a Feedforward Neural Network (FNN) to ensure that h_w stays a contraction map. Each node of the input VNFC graph is replaced by a unit computing h_w for that node, as shown in Fig. 1. When unfolded, each layer i in the encoding network corresponds to an iteration in which the state $s(i+1)$ is computed for each node (VNFC). Each h_w unit thus stores the current state $s_n(i)$ for that node, and when activated, calculates its next state $s_n(i+1)$ corresponding to the next time step. The output of node n is produced by another unit, which implements the g_w, as shown in Fig. 1.

As f_w and g_w are implemented by FNN, the encoding network turns outs to be a RNN. The connections between neurons in the encoding network can thus be divided into internal and external connections. The internal connectivity here would be that within the FNN that represents each parametric function unit; and the external connectivity depends on the topological dependencies from the input SFC topology that determines the connections between individual units.

III. Asynchronous Deep Reinforcement Learning

As demonstrated in Fig. 2, RL follows the Markov Decision Process (MDP) model to train an agent a that observes the state s of the environment at discrete time steps to prescribe actions a that ultimately maximise the reward r, thereby attaining an efficient policy for stochastic scenarios. Q-Learning in this regard is a typical RL algorithm that calculates the value of each state-action pair as a Q-value function $Q(s, a)$. Then, based on the policy (e.g. ϵ-greedy) chooses the action with the largest Q-value, and follows the gradient towards higher rewards. To store the value functions, traditional RL relies on either explicit look-up table (array/hash) or function approximation to store the estimated value functions for each state-action pair. However, RL fails due to dimensionality when dealing with a large state space, as:

- A larger table is not only storage intensive, but also expends more time while traversing the complete range of state-action pairs.
- Linear function approximation is unable to accurately model the estimated value function.

DRL (specifically DQL) overcomes these classical challenges by using Deep Neural Networks (DNN) as non-linear function approximators to learn policy and state-value representations. As demonstrated in [9], by sampling only a fraction of states, neural networks can be trained to fairly accurately approximate the Q-value function. The novel propositions [9] of incorporating 'experience replay' and 'network cloning' have enabled DRL to progress exponentially, making the algorithm powerful enough to beat

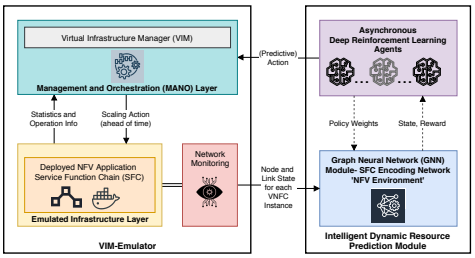


Fig. 4. Proposed system architecture as described in section IV for dynamic resource prediction in NFV environments.

the human world champion in the game of AlphaGo [10].

However, experience replay has several drawbacks. It is computationally intensive on each interaction, and can work with only off-policy algorithms like Q-learning. To this end, a parallel reinforcement learning paradigm seeks to decorrelate the process by asynchronous execution of multiple agents on multiple instances of the environment. These asynchronous methods [11] for DRL seek to replace experience replay, and enable the use of other on-policy RL algorithms as well. Asynchronous DRL stabilizes learning, and reduces the training time that is roughly linear in the number of parallel threads [11].

IV. System Architecture for Dynamic Resource Prediction in NFV Environments

Fig. 4 presents the overall architecture of the proposed system. On a high level, it consists of two primary building blocks– the intelligent dynamic resource prediction module, and the emulation platform for the deployed NFV application SFC. Algorithm 1 presents the pseudo-code for the proposed system architecture, and the linking of components for dynamic resource forecasting.

A. VIM-Emulator

The VIM-Emulator [12] is an emulation platform towards rapid prototyping of network services across multiple points of presence in a cloud environment. In our system architecture, the VIM-Emulator mimics the NFV Infrastructure (NFVI) and Virtualised Infrastructure Manager (VIM). It provides flexible Python API endpoints to control and extend the functionality of all components, which is critical for customizing models and taking charge of the deployment. It offers interfaces that are similar to the control that OpenStack[1] offers, and allows to start, stop and manage VNFs. From the Management and Orchestration (MANO) [2] system perspective, this setup translates to a real world multi-VIM deployment.

[1]https://www.openstack.org/

Algorithm 1: Asynchronous Deep Q-Learning enhanced GNN for topology-aware VNF resource prediction in dynamic NFV environments

Initialise: Weight $\theta = 0$, iteration $i = 0$, max iteration counter T, state $s(i) = 0 \quad \forall n \in N$

procedure OBSERVATION
  Observe f for all VNFC's and their neighbourhoods
end
procedure STATE COMPUTATION
  while $i < T$ **do**
    Compute $s(i+1)$ using equation 3
  end
end
procedure OUTPUT COMPUTATION
  Compute $o(i)$ using equation 2
end

For each deep Q-learning agent thread:

repeat
  Individual agent thread gets initial state s from GNN encoding network formed above
  The agent chooses action a_t with chosen exploration policy based on $Q(s, a; \theta)$
  After executing the action a_t, the agent observes the reward r and a new state s' for the system
  Compute Q^+ next Q-value
  The agent accumulates the gradients over multiple time steps w.r.t. θ
  The agent periodically asynchronously updates the weights on the encoding network
until $i > T$ *(i.e. predefined stopping condition)*;

It is based on Containernet[2], which extends the Mininet[3] emulation framework towards allowing addition and removal of Docker[4] containers acting as compute resources representing each VNF in an SFC during runtime. This is an essential service for NFV based applications and our approach, which requires instantiating or terminating new infrastructure resources for VNFs on the go. It also allows to dynamically change resource limitations of services at runtime, which mimics the real world scenario of fluctuating resource constraints based on network factors. This can be deployed in a custom network topology with realistic network and link characteristics.

To this end, we feed into the VIM-Emulator the topology and data flow dependencies of our emulated SFC. This SFC as shown in Fig. 1 is inspired from the architecture of Project Clearwater[5], which is an an open source implementation of the IP Multimedia Subsystem (IMS) for cloud computing environments. We defined this custom network service via Virtual Network Function Descriptors (VNFD) and Network

[2]https://containernet.github.io/
[3]http://mininet.org/
[4]https://www.docker.com/
[5]https://www.projectclearwater.org/

Service Descriptors (NSDs) [2], [5], [12], and fed this into the emulator as the infrastructure component.

B. Intelligent Dynamic Resource Prediction Module

The SFC topology as shown in Fig. 1 is also input into the GNN module. As also shown in Fig. 1, the forwarding graph topology coupled with the parametric functions map the state of each VNFC node with its neighbourhood, and together form an encoding network to iteratively determine the states and output resource prediction for the VNFCs.

Further, in our prediction module, we propose the use of asynchronous Q-learning to enhance the GNN. This would replace the use of the classic backpropagation algorithm from previous work [6], [7]. The developed GNN encoding network acts as the environment for the deep reinforcement learning agents. Multiple agent threads are executed parallely on the same machine, each with its own view of the environment. These agents spread across the nodes in the GNN encoding network asynchronously observe and feed back the state of the VNFC nodes, which helps in adjusting the weights for the encoding network and reinforcing the learned policy for optimized prediction output. Each agent may have a different exploration policy, which would add diversity to the exploration and further improve the performance. To avoid multiple agents overwriting each others' updates, we accumulate network weight gradients over multiple time-steps before they are applied, as this has been shown to provide some ability to trade off computational efficiency for data efficiency [11]. The reward for the accuracy of output in the encoding network, i.e. a correct scaling decision predicted helps drive the policy to an optimum learned stage. The predicted output is sent to the VIM in the MANO layer of the VIM-Emulator so that scaling action on the actual infrastructure can be taken ahead of time and the resources provisioned can be optimized based on the demand.

V. Conclusion and Future Work

In this article, we propose the system architecture for an Asynchronous DRL enhanced GNN for topology-aware VNF resource prediction in dynamic NFV environments. We use the GNN to model the topological dependencies between the nodes in an NFV SFC, and use the resultant encoding network as the environment for asynchronous DRL agents that help form the policy for predictive scaling of VNFC resources. The asynchronous learning methodology works well towards incorporating the different PoPs of VNFs, and further helps reduce learning and training overheads as compared to classical DRL.

In future work, we plan to share the implementation of the proposed system architecture, and gather the results on how the dynamic resource prediction algorithm best performs. As with any machine learning algorithm, training is a crucial aspect of how the algorithm performs. We intend to test the training phase with different types of data sets, i.e. synthetic, real-world (VoIP traces), and analyze the impact they have on the overall performance. It is also in the pipeline to compare the learning and prediction performance of different DRL algorithms here, and see which one best supplements the GNN model with a good generalisation accuracy. It would also be interesting to see how well the model scales depending on the number of VNFC and VNF nodes in the SFC.

Acknowledgment

This work has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) and is co-funded under the European Regional Development Fund under Grant Number 13/RC/2077.

References

[1] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 236–262, Firstquarter 2016.

[2] 5GPPP, "5g ppp architecture working group: View on 5g architecture." Available Online: https://5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf, Jan. 2019.

[3] R. Mijumbi, J. Serrat, J. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, pp. 98–105, January 2016.

[4] M. Jones, B. Arcand, B. Bergeron, D. Bestor, C. Byun, L. Milechin, V. Gadepally, M. Hubbell, J. Kepner, P. Michaleas, J. Mullen, A. Prout, T. Rosa, S. Samsi, C. Yee, and A. Reuther, "Scalability of vm provisioning systems," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–5, Sep. 2016.

[5] ETSI, "Network functions virtualisation- an introduction, benefits, enablers, challenges & call for action," in *SDN and OpenFlow World Congress*, 2012.

[6] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Transactions on Network and Service Management*, vol. 14, pp. 106–120, March 2017.

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, pp. 61–80, Jan 2009.

[8] K. Khamsi, *An Introduction to Metric Spaces and Fixed Point Theory*. John Wiley & Sons, 2001.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529, Feb. 2015.

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, p. 484, Jan. 2016.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1928–1937, PMLR, 20–22 Jun 2016.

[12] M. Peuster, H. Karl, and S. van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 148–153, Nov 2016.