

Optimised 5G Network Sub Slicing Orchestration (ONSSO) in a Fully Virtualised Smart Company Using Machine Learning

Dr Abimbola Efunogbon

This is a digitised version of a dissertation submitted to the University of Bedfordshire.

It is available to view only.

This item is subject to copyright.

Optimised 5G Network Sub Slicing Orchestration (ONSSO) in a Fully Virtualised Smart Company Using Machine Learning

By

Abimbola Efunogbon
Student Number: **1025306**
May, 2025
Current: MPhil/PhD
Intended: PhD
Mode of Study: Full Time

Word Count: 44954

Supervision Team:

Dr. Enjie Liu
Dr. Renxi Qiu

A thesis submitted for partial fulfilment for the degree of Doctor of Philosophy in the

Institute for Research in Applicable Computing (IRAC)

Faculty:

Faculty of Creative Arts, Technologies & Science

The University of Bedfordshire

Abstract

The traditional method of one size fits all for network resource allocation no longer works for various vertical with the limited available network resource to meet the increasingly rise in demand for diverse requirements to satisfy network quality of service (QoS), quality of experience (QoE) and service level agreements (SLA). The complexity of allocating various network resources to each vertical as required and on-demand has been a major challenge with the flexible 5G network. This research examined existing studies on network slicing, which serves as an enabler for 5G and aim to identify and solve the challenge with efficiently allocating the limited network resource to orchestrate the 5G network slices on-demand. It also investigated the existing challenges of managing current cross domain network resources in an enterprise network with dynamic and unstructured network slice requirements and then proposes a framework to efficiently orchestrate the network slice to meet the varying latency, data rate, mobility, and reliability needs thereby reducing capital and operational expenditure. In this study, the proposed advanced machine learning (ML) algorithm pipeline model designed for optimised sub-slicing of 5G network slice dynamically on demand, re-learning, and service function chaining to orchestrate bespoke network slice is called the Enhanced Subslice Model (eSS). This study further proposes a smart and novel end-to-end adaptive resource management framework which is defined as ONSSO. This proposed framework comprises of functions ranging from the analysis of the diversified network service on a network, to the implementation of the proposed ML-based eSS model which incorporates a series of supervised machine learning algorithms as well as the LazyPredict module to find and suggest the best-fit model, to the making of intelligent decisions for the specific network data environment being monitored in real time or analysed with historical data. It also includes functions ranging from implementing caching to improve the user experience and performance of orchestration right through to employing reinforcement learning techniques for admission control, as well as predicting future network traffic data pattern thereby ultimately adapting slice configurations in real-time and efficiently orchestrating alternative best fit bespoke network slices. This ONSSO is designed to enable and optimise processes within a newly proposed vertical called Enterprise Company Network as a Service (CNaaS) which is based on an industry 4.0 smart company. The integration of the ONSSO into CNaaS provides a scalable and optimised solution to network sub-slicing orchestration in an enterprise network. This CNaaS was ultimately used as a case study in validating the model through simulations conducted using python with MATLAB and Simulink, where different network parameters were analysed under varying conditions. These simulations enabled the identification of optimised resource allocation strategies across multiple slices, considering factors such as latency, bandwidth, and energy efficiency.

Results from the MATLAB simulation demonstrate that the proposed model significantly enhances the flexibility and scalability of network subslice orchestration for dynamic and adaptive network management in future mobile networks. When compared with traditional static orchestration methods, the sub-slicing of the network slice with the proposed machine learning-based pipeline exhibited improvements in resource utilisation, enhancing QoS and QoE for end-users and reduced the risk of denial of service. This research offers a novel approach to network slice orchestration, with potential applications in various sectors such as smart cities and autonomous vehicles. The proposed model not only contributes to the ongoing development of 5G and 6G networks but also provides a foundation for future research on adaptive network management. This research aims to initiate discussion on the new suggested concept of enhanced subslice as well as robotic process automation of network slice orchestration, provide the foundation for the application model which application programming interface can be built upon and spur development on true fully automated self-organising network slice orchestration.

Publications by the Author

Liu, E., Zhao, Y. and Efunogbon, A., 2023. Boosting smarter digital health care with 5G and beyond networks. ITU Journal on Future and Evolving Technologies, 4(1), pp.157-165.

Liu, E., Zhao, Y. and Efunogbon, A., 2023. Medical Imaging and Healthcare Applications Using 5G. In: A. El-Baz and J.S. Suri, eds. Cloud Computing in Medical Imaging. 1st ed. Boca Raton: Auerbach Publications, pp.19-37.

Efunogbon, A., Liu, E. and Qiu, R. (2024) 'Optimal 5G network sub-slicing orchestration in a fully virtualised smart company using machine learning', Future Internet, 17(2), p. 69. Available at: https://doi.org/10.3390/fi17020069.

List of Acronyms

Acronym	Full Form
5G	Fifth-Generation Mobile Network
AutoML	Automated Machine Learning
AutoRL	Automated Reinforcement Learning
CNAAS	Company Network As A Service
eMBB	Enhanced Mobile Broadband
ESS	Enhanced Subslice
mMTC	Massive Machine-Type Communications
NF	Network Function
NS	Network Slice
ONSSO	Optimised 5G Network Sub Slicing Orchestration
SDN	Software Defined Network
URLLC	Ultra-Reliable Low-Latency Communications

Table of Contents

Publications by the Author 3
List of Acronyms 4
List of Tables 12
1.0 Introduction 13
1.1 Current Research Questions 15
1.2 Aims and Objectives 16
1.3 Statement of Proposed Contributions to Knowledge 17
1.4 Structure of the Thesis 19
2.0 Background and Literature Review 20
2.1 Context 20
2.1.1 5G 21
2.1.2 5G Network slicing 22
2.2 Related Work 25
2.2.1 Improving 5G Quality of Service 25
2.2.2 Resource Management in Network Slicing 26
2.2.3 Network sub slicing 27
2.2.4 Network Function Virtualisation (NFV) & Software Defined Network (SDN) 29
2.2.5 Machine Learning 31
2.3 Methodologies 35
2.4 Challenges and Limitations 36
2.4.1 Limited Evaluation Metrics 36
2.4.2 Assumptions and Simplifications in Methodologies 36
2.4.3 Homogeneous Network Slices 37
2.4.4 Scalability and Performance 37
2.5 Future Directions 38
2.6 Summary 38
3.0 Design and Architecture 42
3.1 About The Proposed Model, Framework, Streamlined VNO and Vertical 42
3.2 The 5G Quality Metrics 46
3.3 The Dataset Flow Design 48
4.0 Proposed Vertical Integration – Company Network as a Service (CNaaS) 52
5.0 Proposed Framework – ONSSO 61
5.1 Device Monitoring, Data Discovery, Collection and Preprocessing: 64

5.2 Network Data Collection and Preprocessing: 64
5.3 Network Data Preprocessing and Feature Engineering: 65
5.4 Subslice Feature Engineering 65
5.5 Network Service Type Detection: 66
5.6 Templates: 66
5.6.1 Service Function Templates: 66
5.6.2 Subslice Templates: 66
5.7 Service Function Chaining (SFC) Policy Engine: 67
5.8 Resource Scaling and Allocation: 68
5.9 Service Function Placement 70
5.10 Service Function Chain Configuration 70
5.11 Dynamic Slice Orchestration: 70
5.12 Network Resource Allocation using Machine Learning Algorithm - Reinforcement Learning: 71
5.13 Subslice Optimisation Policies: 71
5.14 Real Time Network Sub Slicing: 71
5.15 Continuous Monitoring of Connected Devices and Learning: 72
5.16 Fault Detection with Behaviour and Security Analysis: 72
5.17 Logging and Dashboard Reporting Capability: 73
5.18 End-to-End Subslice Lifecycle Management: 73
6.0 Proposed Model – eSS 74
6.1 5G Network Subsliced Architecture. 79
6.1.1 Step 1: Identify features or feature selection from the traffic flow dataset using - Dimensionality Reduction Algorithms (PCA, LDA) 81
6.1.2 Step 2: Group features with characteristics using – LazyPredict Algorithms 97
6.1.2.1 98
6.1.2.2 100
6.1.2.3 101
6.1.3 Step 3: Group applications or sevices – New application service classification 107
6.1.3.1 108
6.1.3.2 108
6.1.4 Step 4: Understand trend for classification using – Explanatory Algorithms (SHAP or LIME) 109
6.1.5 Step 5: Define each slice and allocate to grouped applications 123

6.1.6 Step 6: Auto assign network slice to new setvices with similar features/characteristics using - Similarity Algorithms (KNN, SVD, Cosine, Euclidean Distance, Levenshtein, Jaro-Winkler, and other classification machine learning models) 124
6.2 Proactive and Adaptive Network Slice Prediction 124
6.3 The Service Function Chaining Enabled Domain Layer 126
6.3.1 Communication, Slice Requirement and KPI 129
7.0 Proposed Framework Algorithm 130
7.1 Digital Twin 131
7.1.1 Introduction 131
7.2.2 Case Study: Digital Twin-Based CNaaS in 5G Network Subslice Orchestration 131
7.2.3 Simulation Setup 133
7.2.4 Experimental Results and Analysis 134
7.2.5 The Digital Twin Function Code 135
8.0 Evaluation 138
8.1 Simulation Setup 138
8.2 Dataset Evaluation 139
8.2.1 Result 143
8.3 Network Slice QoS Metrics Evaluation 144
8.3.2 Result 146
8.4 Optimised Number of Slice Selection 146
8.4.1 Result 146
8.4.1.1 The Elbow Plot 146
8.4.1.2 The Silhouette score 147
8.4.1.3 The Davies-Bouldin Index 149
8.5 Slice Allocation 150
8.5.1 Analysis for Each Slice Network Functions Requirements: 151
8.5.2 The Individual Metric Importance and Result 154
8.5.2 Result 163
8.6 LazyPredict Evaluation 164
8.6.1 Implementing Naïve Bayes 164
8.6.2 Implementing Linear Discriminant Analysis 164
8.6.3 Implementing Logistic Regression 165
8.6.4 Implementing Random Forest Classifier 166
8.6.5 Result 166
8.7 Varying Dataset Input in eSS Model 167

8.7.1 Part1 Dataset 167
8.7.2 Part2 Dataset 171
8.7.3 Part3 Dataset 173
8.7.5 Result 176
8.8 Resource Utilisation 177
8.8.1 Result 178
8.9 Summary 178
9.0 Conclusion and Future Work 180
9.1 Summary of Contributions 180
9.2 Key Findings and Contributions 180
9.2.1. Proposed Adaptive Framework 180
9.2.2. Machine Learning Model (eSS) 180
9.2.3. A New Vertical (CNaaS) 180
9.2.4. Streamlined Mobile Network Operators Collaborative Process 181
9.2.5. Integrated Auto-Selection Best Fit Machine Learning Model: 181
9.2.7. Scalability and Adaptability 181
9.2.8. Efficiency Metrics 181
9.3 Implications of the Findings 182
9.3.1 Network Efficiency: 182
9.3.2. Operator Flexibility 182
9.3.3. Cost Reduction 182
9.3.4. AI in Telecommunications 182
9.4 Limitations and Future Work 182
9.5 Concluding Remarks 183
References 184
Appendices 190
Appendix A: Ethics Form 190
Appendix B: Research Data Management Plan 195
Overview 195
Compliance 195
Sharing data – see 'Preparation of Data for Sharing and Archiving' in the RDM Checklist 197
Implementation – see 'Responsibilities' in the RDM Checklist 197
Appendix C: Source Code 198

List of Figures.

Figure 1.3.1: Relationship between eSS and ONSSO and CNaaS 19
Figure 2.1.1.1. 5G challenge, facilitators, and design fundamental [21] 22
Figure 2.1.1.2. 5G System Components 22
Figure 2.1.2.1: Network Slice Components 24
Figure. 2.2.4.1. Mapping EPC functionalities to VNFs 30
Figure 2.2.5.1. Classification of machine learning techniques [23] 32
Figure 3.1.1: NS Orchestration Cycle From Application Layer to Infrastructure Layer 44
Figure 3.1.2: 5G Network Slicing Process Design 44
Figure 3.1.3: Network Subslice Orchestration Methodology 45
Figure 3.3.1: Network Slice Orchestration Flow from Data Collection 49
Figure 3.3.2: Network resource management and sharing architecture [45] 50
Figure 4.2: Company and its Departments overview 54
Figure 4.3. Smart Company Structure 56
Figure 7.3: Slice lifecycle 58
Figure 4.5: Process Flow Architecture 60
Figure 5.1.1 Device Monitoring, Data Discovery, Collection and Preprocessing 64
Figure 5.2 Network Data Collection and Preprocessing 65
Figure 5.3 Network Data Preprocessing and Feature Engineering 65
Figure 5.4.1 Subslice Feature Engineering 65
Figure 5.4.2 Subslice Feature Engineering 65
Figure 5.5 Network Service Type Detection 66
Figure 5.6.1 Service Function Templates Pseudocode 66
Figure 5.6.2 Subslice Default Template 67
Figure 5.7.1: Service Function Chaining (SFC) Policy Engine Pseudocode 68
Figure 5.8: Resource Scaling and Allocation Pseudocode 69
Figure 5.9 Service Function Placement Pseudocode 70
Figure 5.10 Service Function Chain Configuration Pseudocode 70
Figure 5.11 Dynamic Slice Orchestration 71
Figure 5.12 Network Resource Allocation using Machine Learning Algorithm - Reinforcement Learning 71
Figure 5.13 Subslice Optimisation Policies 71
Figure 5.14 Real Time Network Sub Slicing 71
Figure 5.15 Continuous Monitoring of Connected Devices and Learning Pseudocode 72
Figure 5.15.1 Fault Detection with Behaviour and Security Analysis 72
Figure 5.15.2 Fault Detection with Behaviour and Security Analysis 73
Figure 5.17 Logging and Dashboard Reporting Capability Pseudocode 73
Figure 5.18 End-to-End Subslice Lifecycle Management Pseudocode 73
Figure 6.0.1: Types of Application Services on Smart Company Network 74
Figure 6.0.2: Network Sub Subslice Structure. 75
Figure 6.0.3. System architecture diagram showing the components and interactions. 76
Figure 6.0.4. Smart Company End-to-end network slice overview. 77
Figure 6.0.5: API call and response to instantiate network slice instance 79
Figure 6.1.1: The 6 Steps Cycle of eSS Framework 81
Figure 6.1.1.1: Unused Features 81

Figure 6.1.1.1: Import libraries. 82
Figure 6.1.1.3: Top 5 data returned by the head () 83
Figure 6.1.1.4: All data features available in dataset. 83
Figure 6.1.1.5: Make data numerical. 84
Figure 6.1.1.6: Converted Dataset into Numerical 85
Figure 6.1.1.7: Import libraries for scaling data and separating the application reference point. 85
Figure 6.1.1.8: Data split for training and testing model 86
Figure 6.1.1.9: Fitted training model and transformed dataset. 86
Figure 6.1.1.10: Number of records in training and testing model 86
Figure 6.1.1.11: PCA application on the training model 86
Figure 6.1.1.12: Output showing the number of features. 87
Figure 6.1.1.13: Extract important feature names. 87
Figure 6.1.1.14: Selected 33 Principal Components 88
Figure 6.1.1.15: Histogram of selected features 89
Figure 6.1.1.16: Code for scree plot 90
Figure 6.1.1.17: Scree Plot 90
Figure 6.1.1.18: Plotting the Eigenvalues against the number of features. 90
Figure 6.1.1.19: Scree plot of Eigenvalues against the number of features 91
Figure 6.1.1.20: Explained variance ratio against principal components 91
Figure 6.1.1.21: Boxplot of all features against ProtocolName 94
Figure 6.1.1.22: Correlation matrix 95
Figure 6.1.1.23: Scatter plot of two features 96
Figure 6.1.1.24: PC1 against PC2 96
Figure 6.1.2.1: K-Means clustering. 98
Figure 6.1.2.2: Results after comparing various machine learning models' performance with training the dataset 99
Figure 6.1.2.3: Number of active application services on company's network 101
Figure 6.1.2.4: Validating Models – Random Forest 102
Figure 6.1.2.5: Precision and Recall Score – Random Forest 103
Figure 6.1.2.6: Validating Models - Decision Tree Classifier Model 103
Figure 6.1.2.7. Precision and Recall score – Decision Tree Classifier Model 103
Figure 6.1.2.9: Precision and Recall Score - K-NN 104
Figure 6.1.2.10: Classification report – K-NN 104
Figure 6.1.2.11: Confusion Matrix – K-NN 106
Figure 6.1.2.13. Results for K-NN 106
Figure 6.1.2.14: Plot tree to chart the decision structure for the Random Forest classifier. 107
Figure 6.1.4.1: Code for generating the global variable importance. 109
Figure 6.1.4.1: Global variable importance 110
Figure 6.1.4.3 Application Type Mappings with Default Network Characteristics Mappings 113
Figure 6.1.4.4 Resource Alternative Selection Algorithm 116
Figure 6.1.4.1.1: SHAP code for variable importance plot 117
Figure 6.1.4.1.2: Variable importance plot 118
Figure 6.1.4.1.3: Correlation – SHAP variable importance plot 118
Figure 6.1.4.1.4: Correlation – Simplified version 119
Figure 6.1.4.1.5: Dependence plot 119
Figure 6.1.4.2.1: LIME to explain the model and predictions for Windows Update service. 120

Figure 6.1.4.2.2: LIME Notebook report for Windows Update service interpretability 121
Figure 6.1.4.2.3: Total Coefficient 121
Figure 6.1.4.2.4: First two application service records predicted of the test dataset. 121
Figure 6.1.4.2.5: List of LIME coefficients 122
Figure 6.1.4.2.6: LIME to explain the model and predictions for YouTube service. 123
Figure 6.1.4.2.7: LIME Notebook report for YouTube service interpretability 123
Figure 6.1.6.1: Network Slice Layer – Enhanced Subsliced Network 124
Figure 7.2.1: Proposed Digital Twin Architecture Integrated with CNaaS for Subslice Orchestration 132
Figure 7.2.4.1: Speed of Orchestration Decision-Making 134
Figure 7.2.4.2: Quality of Orchestration Decisions Over Time 135
Figure 8.1.1: Simulation experimental setup 138
Figure 8.2.1: Stratified Splitting of the Dataset 139
Figure 8.2.2: Dataset Split – part1 141
Figure 8.2.3: Dataset Split – part2 142
Figure 8.2.4: Dataset Split – part3 142
Figure 8.2.5: Class Distribution 142
Figure 8.3.1: Calculated Mean for Each Metric Grouped by Application 144
Figure 8.3.2: Network Slice Categorisation 146
Figure 8.4.1: Elbow Plot to Determine Number of Slices 146
Figure 8.4.2: Silhouette Score to Determine Number of Slices 147
Figure 8.4.3: Davies-Bouldin Index to Determine Number of Slices 149
Figure 8.5.1: Slice Centers and Allocation 150
Figure 8.5.2: Applications And Slice Distribution In Dataset 152
Figure 8.5.3: Applications Per Slice 153
Figure 8.5.5: Metrics Decision Importance 155
Figure 8.5.6: Alternative Network Slice 155
Figure 8.5.7: Standard and Bespoke 5G Slice 156
Figure 8.5.8: Distribution of Network Slice Features Visualisation With Boxplot 157
Figure 8.6.1.1: Naïve Bayes 164
Figure 8.6.1.2: Linear Discriminant Analysis 165
Figure 8.6.1.43: Logistic Regression 165
Figure 8.61.4: Random Forest Classifier 166

List of Tables

Table 2.3.1. Highlights of existing studies researched 39
Table 4.1. Sample Departments with Network Slice Requirements 58
Table 6.1.1: Transport and core network of CNaaS 79
Table 6.1.2.1: Results from LazyPredict module showing machine learning models performance with training the dataset 100
Table 6.1.2.2: Classification Report – K-NN 104
Table 6.1.3.2.1: Extracted network slice characteristics and measurement scale 108
Table 6.1.4 Application Types with Fall Back Subslice Feature 111
Table 7.0.1: Proposed Framework Algorithm 130
Table 8.2.1. Dataset distribution - Number of Rows 142
Table 8.2.2. Performance Evaluation Result of the Dataset Types 143
Table 8.3.1. Typical Classification of Latency, Bandwidth and Jitter Metric Thresholds 145
Table 8.4.1. Results Of Charts for Number Of Slices Selection 149
Table 8.5.1. Characteristics of 5G Standard Slice 162
Table 8.5.2. Low, Medium, High Categorisation of Network Slices 163
Table 8.6.1.1. Summary LazyPredict Validation 166
Table 8.7.8.1. Summary Model Accuracy 176

1.0 Introduction

We are constantly being faced today with the rise in the need for better technology. Optimised, available, scalable, reliable, and generally good quality technologies are the basic requirements in every vertical today socially, technologically, environmentally, politically, and economically. The telecommunications industry is undergoing a transformation with the rollout of the fifth generation (5G) cellular networks and the ongoing research into the sixth generation (6G) cellular networks technology and beyond. These advancements promise to transform mobile broadband communication delivering unprecedented quality of service, connectivity, and flexibility in mobile networks [76] requiring networks to be more adaptable, efficient, and scalable than ever before. A critical enabler of this flexibility promised by 5G is network slicing, which allows operators to create virtualised and independent slices of the network, each optimised for specific performance requirements. It enables the paradigm shift in mobile broadband communications with its ability to create multiple virtual verticals from end to end while sharing the available physical infrastructure [77]. These slices can deliver customized services for specialised applications as well as within the three primary category of slices defined by 3GPP which are enhanced mobile broadband (eMBB) for high-definition streaming, ultra-reliable low-latency communication (URLLC) for autonomous systems and Massive Machine-Type Communications (mMTC) for massive number of connected devices that send small amounts of data sporadically, like sensors and IoT devices.

Orchestrating a whole standard slice with all the required resource to configure that slice might result in over provisioning and allocating more resource than is required by the user equipment. This could increase the risk of denial of slice for other user equipment due to the limited available physical resource. Efficiently provisioning a network slice to meet the varying network traffic patterns and fluctuating service demands requires smart and intricate decisions regarding the allocation of the limited available network resource including the physical infrastructure, scaling and isolation of the network functions [78]. Network sub-slicing further divides each network slice into granular subslices which are dedicated to specific service, tenant, or application instance. This manages the available network resource and allocates only what is required to each service or user demanding a network slice. This helps reduce the capital expenditure cost which was identified as a current 5G network slice orchestration challenge. Each orchestrated network subslice is configured to meet specific user equipment's network requirement. The network subslice is made up of network functions which are chained together to form end-to-end network slice. The ability to define, reorganise, share, and chain these specific required network functions enables optimised allocation of resources and allows for high flexibility and customisation.

With the increasing complexity of modern networks and the recent advancements in artificial intelligence, organisations are significantly considering the optimisations of their business models to enable low cost, scalability, adaptability, availability, and flexibility with optimised performance and the need to run complex applications, data analysis amongst other use cases as the shift to the use of technology with already limited resource is on the rise. Addressing this challenge requires a more dynamic and intelligent approach to streamlining slice orchestration workflow and orchestrating network slices, capable of optimizing resource distribution based on live network conditions. This research proposes an architecture that uses a message bus which all network slice service providers subscribe to. This grants service providers the ability to analyse and be aware of the predicted network resource need in real-time. Network slice orchestration workflow is streamlined thereby reducing latency. Network providers such as Virtual Network Operators (VNO), Mobile Network Operators

(MNO), Infrastructure Providers and Application Service Providers (ASP) will be able to seamlessly collaborate and see end-to-end user equipment service request requirement in real time. The traditional static methods of orchestrating network slices have been proven to be inadequate. Static approaches rely on pre-configured settings that cannot adapt to real-time changes in network conditions or varying service demands. This results in inefficient resource allocation, network congestion, and poor Quality of Service (QoS) for end-users.

One of the main technologies referred to as a Green IT which drastically helps reduce operation and infrastructure costs is Virtualisation [18]. Studies have shown that virtualising part of the infrastructure such as the operating system, the servers, the storage, or even the entire IT infrastructure dramatically reduces cost and down time. Virtualization effectively manages available infrastructure such as servers, increases flexibility and generally effectively reduces system complexity [19]. Efficiently orchestrating a 5G network slice on demand in this study leverages virtualisation technology on which software defined network (SDN) is developed.

With the shift from application first to technology first, there is increase in unknown networks with varying and unstructured network slice demands. The complexity of efficiently orchestrating the network slice to meet the growing demand remains a challenge [1]. The currently available network slice for each vertical is not only limited but also needs to be efficiently and efficiently orchestrated to improve quality of service, quality of experience and improve availability. This is a major challenge because having low latencies when running critical applications or having the right amount of bandwidth even for social activities such as streaming videos while mobile is critical to making the technology a success and eases the development of new applications that uses the new technology. Whilst one challenge is the limited availability of the network resources, another challenge faced is the complexity in management of the available limited network resources to service the increasingly diverse network slices quality of service requirements in a vertical [1]. Smart products, smart factory, smart devices, and smart communications between human to human, human to machine and machine to machine all require network resources which the 4G and all previous generation network do not have the capacity to satisfy the requirements efficiently. An one size fits all will not efficiently meet all the diverse requirements by the various aspects of these applications. 5G technology however promises to fulfil the network flexibility requirements of these applications. With the myriad of advancements introduced by 5G technology, it hopes to meet the dynamic demand of network slices with its introduction of low latency, higher data rates, massive machine connectivity and increased reliability. Recent advancements in artificial intelligence, particularly machine learning (ML), also offer promising solution by allowing networks to predict traffic patterns, adapt resource allocation dynamically, and learn from historical data to improve decision-making.

Leveraging these capabilities, this study proposes a novel beyond Industry 4.0 end-to-end new vertical called CNaaS framework to efficiently orchestrate 5G network slice over a whole network of collocated departments or distributed company department networks in real time, thereby improving the Quality of Service (QoS) and Quality of Experience (QoE) across the entire company networks. As part of this proposed new vertical, this study explores the use of a series of machine learning algorithm to formulate a ML-based pipeline model called eSS for adaptive network subslice orchestration in 5G and beyond. Specifically, it allows network slices orchestration to adjust automatically to shifting demand and optimize the use of available resources. The proposed novel ML-based pipeline model effectively subslices the network slice in real time to use only the required network function or suggest network functions that can still meet the user needs but with much reduced resource allocation depending on

network resource availability and dynamic user requirements. This chains the defined network functions and efficiently orchestrates a 5G network slice. Our framework leverages LazyPredict which is a Python module of a series of supervised machine learning algorithms. The supervised algorithm consists of both regression and classification machine learning algorithms which dynamically choose the best-fit algorithm for a new unknown environment. LazyPredict simplifies the process of training multiple machine learning models. It is lightweight however could require significant computational resources for particularly large datasets. In this study, Although Google Colab, which is a free cloud service that provides access to computational resources was used to leverage scalability with the following specifications: a dual-core Intel Xeon CPU, 2.3 GHz, Access to Nvidia GPUs - T4 and approximately 12 GB of RAM for testing, the architecture is designed to be flexible and scalable across different environments, ranging from local machines to high-performance clusters. The computational resource requirements and computation time for the proposed architecture are primarily dictated by the complexity of the tasks being performed and the scale of the problem for improved performance, we recommend the following hardware specifications:

- CPU: Quad-core or higher, 3.0 GHz or faster (e.g., Intel Xeon, AMD Ryzen)
- GPU: NVIDIA Tesla P100, T4, V100, or A100
- RAM: 16 GB or more

Utilising these specifications can significantly reduce processing time and improve the efficiency of model training and evaluation. Additionally, reinforcement learning (RL) techniques is employed to make intelligent predictions accurately on future network traffic pattern and make decisions on slice resource allocation on demand. This forms part of the proposed Enhanced Subslice (eSS) module. The proposed eSS module also finds the best alternative fit network slice thereby ensuring all user devices get a network slice that meet their needs. The module also incorporates the concept of cloud computing for scaling to either pay for more resources or configure to use the default alternative best fit only. This caters for proactive scaling of slice resources to meet anticipated demands.

Through a series of simulations conducted, this study extensively evaluates the performance of the proposed model under various network conditions. These simulations provide insights into significant improvement in key performance metrics such as latency, bandwidth, throughput, jitter, packet loss and resource utilisation when comparing the machine learning-based approach with conventional static methods. Explainable AI is showcased in all aspect of the vali-dation. By demonstrating the advantages of adaptive orchestration, this research contributes to the growing body of work on intelligent network management systems for next-generation networks. As the telecommunications industry continues to evolve, the integration of machine learning into network orchestration may prove essential for meeting the growing demands of emerging technologies.

1.1 Current Research Questions

1. How can the current challenge of 5G network slice orchestration optimisation be minimised primarily in an extreme and real-world scenario. How can the 5G network slice be efficiently orchestrated in a dynamic network and unknown environment thereby formulating a novel completely smart company based on the fourth industrial revolution (industry 4.0) featuring Internet of Things applications (IoT) and other diverse artificial intelligence as well as standard applications over the company's network?

2. Can an end-to-end 5G network slice orchestration be smart and fully intelligent with little or no human assistance which will be reconfigurable and leverage on the self-organising network technology?
3. Can the three main 5G network service slice types identified by 3GPP SA2 TS23.501 [66] be further subsliced into more granular subslice of logical network slices to help deliver various services with improved Quality of service and user experience in order to achieve the 5G network communication specification of flexibility and programmability with the use of machine learning?
4. What key features can be considered for categorising the subsliced network slice to manage the available network slice efficiently by using Generative AI?
5. What framework can be proposed for efficiently slicing a dedicated network end to end for applications with multiple and diverse requirements?

1.2 Aims and Objectives

This research aims to manage the limited available resources proactively and efficiently in a 5G network slice for any vertical application in an unknown environment which may possibly include IoT applications as well as diverse devices or applications, thereby making the architecture and implementation flexible and customisable. It aims to provide a framework to meet the specific network slice needs of diverse user applications on the network dynamically in real time.

To achieve these aims, the following objectives would help in facilitating this:

1. Conduct an in-depth review of existing studies on efficiently and intelligently orchestrating 5G network slice, paradigm shifts, major milestones, and the associated research direction highlights.
2. Research and identify the key effective categories on which to base the network subslice on to efficiently manage limited resources on demand, with improved Quality of Service and Quality of Experience.
3. Propose a machine learning based vertical as a reconfigurable framework which is fully intelligent, optimised and based on self-organising network (SON). A framework that encompasses data collection and analysis, resource management with the enhanced option of the use of service functions to dynamically choose alternative template based on available resources, policy enforcement, and service function chaining.
4. Analyse the subslice metrics which efficiently yields QoS as well as QoE and propose a machine learning algorithm pipeline model for a new subslice layer classification which is proposed as Enhanced Subslice (eSS) to predict the optimised slice for an efficient and high-performance network slice in the system. The algorithm will be selected based on tests and evaluation of Generative AI models and various machine learning models.
5. Ultimately propose an ONSSO framework which encompasses the both the proposed CNaaS vertical framework and eSS model used in the framework. The ONSSO framework incorporates self-organising network capabilities with service functions to dynamically and efficiently allocate network resources in real-time while continually monitoring each phase within the framework end-to-end.
6. Evaluate and present a prototype of the vertical application in the form of a digital twin company as a case study for the proposed service using MATLAB.

1.3 Statement of Proposed Contributions to Knowledge

This thesis contributes to the existing body of knowledge on efficiently orchestrating and allocating available 5G network resources in any known or unknown applications running diverse types of applications while automating the process end to end using machine learning. The main contributions of this study can be highlighted as below:

- **Optimised Network Slice Allocation Using Real-Time AutoML Model Comparison:** Optimised network slice resource allocation has been a major challenge in 5G network communication especially with the uncertain environment and increasing diverse network resource requirements by new and existing verticals. Numerous studies are being carried out to address this challenge, however, existing approaches often employ various methods that remain costly and inefficient. Machine learning is also being employed in recent studies however most of the previous studies have selected algorithms either based on known effectiveness of a particular algorithm in solving certain problems or by comparing two or three algorithms. These studies then propose an efficient one based on the result of the performance. This research will be using a model that automatically selects the best classification machine learning algorithm that best fit the data inputted. In this research, the LazyPredict machine learning algorithm will be used to analyse the dataset with over twenty machine learning algorithms and results compared to pick the algorithm with highest accuracy as well as speed. These machine learning algorithms will range from set of supervised, unsupervised to reinforcement learning model. More machine learning algorithms can be added to the LazyPredict if desired to increase the number of algorithms compared as best fit for the network data captured.
- **Subslice Features and Metrics for Optimised Network Slice Orchestration:** The three main types of 5G network service slices defined by 3GPP which are enhanced mobile broadband, ultra-reliable low latency communication and massive machine type communications are too broad and sometimes provide more resources than required by some services on the network. There is need to reduce operating cost by not paying for what is not required in terms of network slice and to reduce capital expenditure by optimising the limited available network resource. Research has been carried out to slice these main slice types into three subslices namely: platform-based service, application-based service and infrastructure-based service. Sub slicing these standard virtual networks allows for the standard slices to be orchestrated to meet the specific needs of services and applications. It allows for the customisation of network functions which optimises available resources within a network slice and enables the ability to further provide differentiated and specialised services to diverse applications and user groups efficiently. This research identified granular features to further subslice the 5G network slices into using Generative AI, thereby forming the required network function by each dynamic network service. This will then be used to formulate the enhanced subslice application model (eSS) proposed in this study. This eSS model optimises cost and efficiently orchestrate the available network resources, allocating only the necessary resources and ensuring that services pay only for what they use. This makes it easy to activate and deactivate network slice requests, as well as reallocate slices as needed.
- **5G Enterprise Network as a Service:** With the availability of 5G and beyond, new applications are increasingly being deployed in an unknown and uncertain environment with dynamic requirements across multiple infrastructures or domains. Augmented reality, virtual reality, driverless cars, massive internet of things sensors and similar applications are demanding for

diverse network slice requirements. This research proposes a new vertical framework to efficiently orchestrate network slice by sub-slicing into more categories than previously studied to improve network slice performance and efficiently utilise the network resource available with the use of machine learning algorithms in a proposed enterprise network case study. Company Network as A Service vertical was formulated as a framework to efficiently orchestrate network slice resource in a cross-domain network with changing network slice requirements.

- **Enhanced Subslice Machine Learning Pipeline Model:** Data packet flow was analysed using machine learning in this study and this was used to identify the key features that can be used to further subslice the 5G network slice and how they could be interpreted with application programming interface (API) to form the network slice template for initiating a network slice instance. These steps form the foundation of the formulated algorithm which the application model called eSS proposed in this study is based on.
- **Self-Organising Network and Service Function Chain as Part of Designed Framework:** An innovative, novel, and comprehensive subslice orchestration framework that incorporates machine learning with service function chaining and the self-organising network capabilities is being modelled to advance the management of 5G networks. The proposed framework in this research aims to incorporate self-organising capabilities within the dynamic 5G networks environment and complements the autonomous network function to dynamically, efficiently, and efficiently orchestrate a 5G network subslice.
- **Compliance and Policies Enforcement as Part of Designed Framework:** This research aims to shed lights on the challenges and difficulties of 5G network security and compliance and incorporates the development of the security and access policies within the orchestration framework. The framework also enforces these security and compliance requirements taking them into consideration while efficiently allocating network resources. This is expected to contribute to the existing knowledge base of 5G network policy management,
- **Alterative Network Function Chaining**: This research also aims to use tailored service function templates to provide standardisation of 5G network subslice and to efficiently allocate resources if the initial resource requested for is not enough. The service function can either scale or intelligently choose the alternative slice template that will meet the subslice requirement. The selection of alternative best fit slice in the event of limited resources for the best fit slice as well as its orchestration is implemented in this research's proposed machine learning pipeline.
- The proposed ONSSO framework which include the CNaaS and the eSS is expected to be validated through simulation with the use of generative AI and offer empirical evidence of its efficiency and effectiveness. This will contribute to the knowledge of autonomously orchestrating 5G network slices.

The relationship between the proposed ML-based pipeline model, eSS, for resource allocation and the proposed adaptive framework, ONSSO, for end-to-end network slice orchestration and the enablement of a new vertical, CNaaS as well as a streamlined mobile network operator is shown in Figure 1.3.1. The proposed novel machine learning-based framework addresses the challenge of 5G network optimised network slice orchestration and a novel ML-based model pipeline addresses resource allocation. These then enable CNaaS and a better efficient process for mobile network operators to access real-time slice request information.

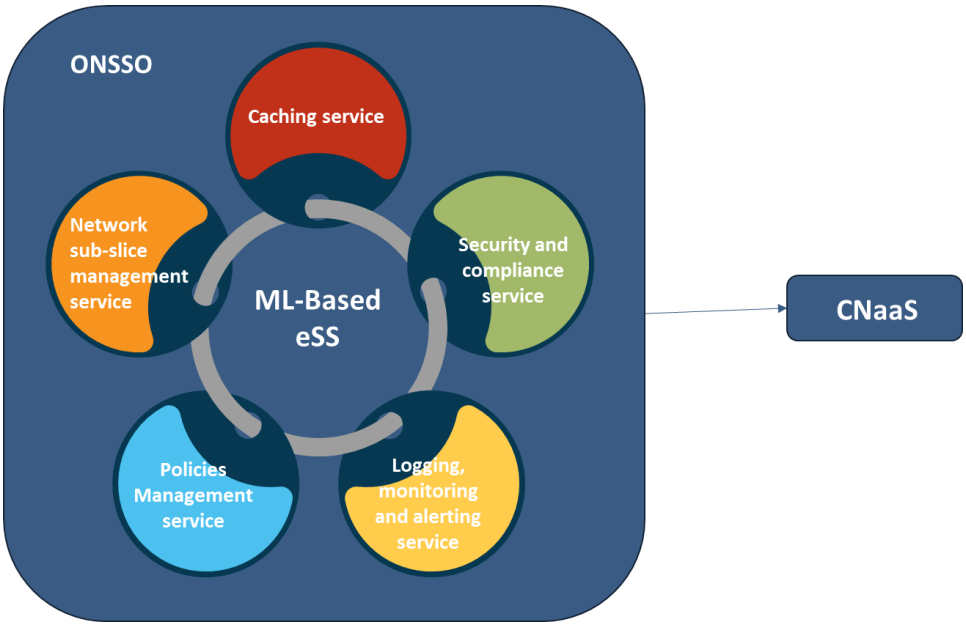


Figure 1.3.1: Relationship between eSS and ONSSO and CNaaS

1.4 Structure of the Thesis

This thesis is organised into 9 main Cchapters and structured as follows:

- Chapter 2 delved into the concept of network slicing in 5G and explored technologies leveraged on in this study. Chapter 2 then further studies the background of the enabling technologies and the existing related literatures which give an insight to the various technologies that enable network slice orchestration and provides a comprehensive overview.
- Chapter 3 discussed the design, architecture and methodology employed in this study.
- The proposed novel vertical integration (CNaaS), the end-to-end self-organising integrated framework (ONSSO), and the machine learning based application model (eSS) were then discussed in Chapters 4, 5 and 6 respectively.
- Chapter 7 described the algorithm for implementing the framework using application programming interface as well as a digital twin.
- The results and observations from the experiments carried out to validate the framework and model were discussed extensively in Chapter 8.
- Chapter 9 concludes the thesis by presenting the potential future work and highlighting the contributions made.

2.0 Background and Literature Review

2.1 Context

Emerging technologies are disrupting the communication world. We are constantly seeing technology advancements giving rise to the daily increase in the number of people getting attracted to the various technology-inspired applications every day and unique ideas generated daily based on intelligent implementation of available technology giving users quality personalised customer experience and service. We are constantly seeing growth in demand for existing wireless internet use and the strain on the available bandwidth is now becoming obvious as the available network resources is not capable of meeting the type of agile resource which is flexible and low cost required by these new generation applications [12].

The previous mobile communication technologies, including the current one all have a common feature as they evolve, an increase in available bandwidth and data rate. With the increasing number of applications that require low latency and high data rate today, the fifth generation (5G) mobile technology offers to meet these demands. It promises to provide seamless high data rate service for applications that especially require wide area coverage with high speed, high hotspot for applications running on edges that require it, connect multiple sensors together seamlessly, provide low latency and high data rate for various verticals that require this such as smart ambulance and driverless cars [1].

5G mobile networks proposed to cater for diverse verticals thereby giving network providers the opportunity to provide bespoke network configuration and slices to various vertical industries satisfying their network service delivery needs, multi service needs and multi tenancy support [38].

The promises of 5G networks and requirements have given rise to the exponential growth in the number of diverse verticals requiring ultra-high speed data transmission of about 10Gbps or ultra-low latency of about 1ms resulting in needs of heterogenous networks and ultra-dense networks with speeds more than 1000 times the known 4LGE network and about 5 times lower in latency [23]. Smart and connected devices have increased in the number of data shared between devices hence requiring a reliable and stable network slice to meet these diverse and unique requirements.

The demand for these diverse network slice requirements on different verticals has made it challenging to meet the needs of each application and challenging to meet the 5G feature requirements. The available physical resource block is limited which need to be shared among subscribers thereby increasing the cost due to high demand but low availability. This results in bad network coverage, applications on queue to receive requested network slice and sometimes denial of request due to need of making policies to prioritise slices. Especially, the network slice management and allocation have been researched but remain a major challenge today for both users and providers.

In this Chapter, this study will discuss the existing background knowledge on 5G technology in relation to meeting the requirements and demands required for efficiently allocating the 5G network slice resource. In addition, this Chapter will review the existing studies on 5G network sub slicing which is a concept that enhances the resource allocation thereby allowing for a more granular customisation, improved isolation, and enhanced flexibility to meet the specific needs of diverse user needs. This review and background knowledge provides the answer to the question, why demand for 5G is on the rise and challenges faced. The background of other enabling and related technology such as network

slicing, network function virtualisation, machine learning and software defined network studies in relation to 5G is also discussed this in this Chapter.

2.1.1 5G

The evolution of mobile technology since inception has been remarkable and with all the developments and deployments, there was a need for standardisation and maintenance which gave birth to the 3rd Generation Partnership Project to satisfy this requirement. According to the 3rd Generation Partnership Project (3GPP) website, it was established in 1998[13] with the purpose of developing standards and specifications for the 3G mobile phones at the time. This was based on the Global Systems for Mobile Communications (GSM) which specifies the protocol for 2G and 2.5G; the second generation of mobile cellular networks [14]. This has since evolved and now provides specifications not only for Universal Mobile Telecommunications System (UMTS) which specifies standard for the third-generation digital cellular networks (3G) but also for Long Term Evolution (LTE), LTE Advanced, LTE Advanced Pro, 5G and 5G NR which provides technical specifications and reports for these beyond mobile technologies [14].

With 5G being actively researched and worked on by the working groups (WGs) and technical specification groups (TSGs), it promises to provide more than just high-speed internet. Release 16 in 2020 by 3GPP addresses the standardization of 5G [15] The way we use technology or live is changing every day. Just as the focus shifted from business first to customer first, so has the release of next generation technology shifted from technology first (while companies figure out what to do with it) to application first and then technology built to meet the needs. 5G promises to bring transformative impact to the world of mobile communication technologies [15]. This 5G technology with its ultra-reliable low latency communications (URLLCs), enhanced mobile broadband (eMBB), Tactile Internet (TI) and massive machine-type communications (mMTCs) features comes with its own challenges which are already surfacing with the massive amount of data being generated and consumed in the massively connected networks [15]

With the amount of data being consumed and shared as more devices get connected taking advantage of this revolutionary technology, there is a need for not only privacy and security of data but also for improved user experience and optimised management of the limited available network resources. This has been possible with the technology shift towards Software Defined Network (SDN), Network slicing, Network Function Virtualisation (NFV), Massive Multiple Input Multiple Output (mMIMO), Non-Orthogonal Multiple Access (NOMA), ultra-dense small cell network, Device to Device (D2D) and Machine to Machine (M2M) communications [15]. However, the full deployment of 5G end to end and orchestration of the network slice to help provide the sort of performance efficiency 5G promises still requires lots of improvements and faces challenges as shown later in this study.

The study in [1] also went on to show that to reduce expenditure and improve quality of service due to the limited available network resources in 5G, network slicing was being studied to enable physical networks to be split into multiple logical networks. Each can then be customised based on the characteristics of the network required. While this contributes to the concept of network slicing, it does not address the challenge of efficiently orchestrating these logical networks. The concept of improving mobility management was studied in [1] to propose an architecture for 5G systems based on network slicing.

Today's organisations communication network needs are changing and existing 4G or prior mobile communication networks are not meeting these demands or capable of solving them. A. Gupta et

al.[20] has highlighted the six main challenges for today's organisations and vertical applications that 5G technology proposes to solve. These are cost efficient and improved quality of experience provisioning, smooth massive device connectivity, higher capacity, very low end to end latency and higher data rate [2].

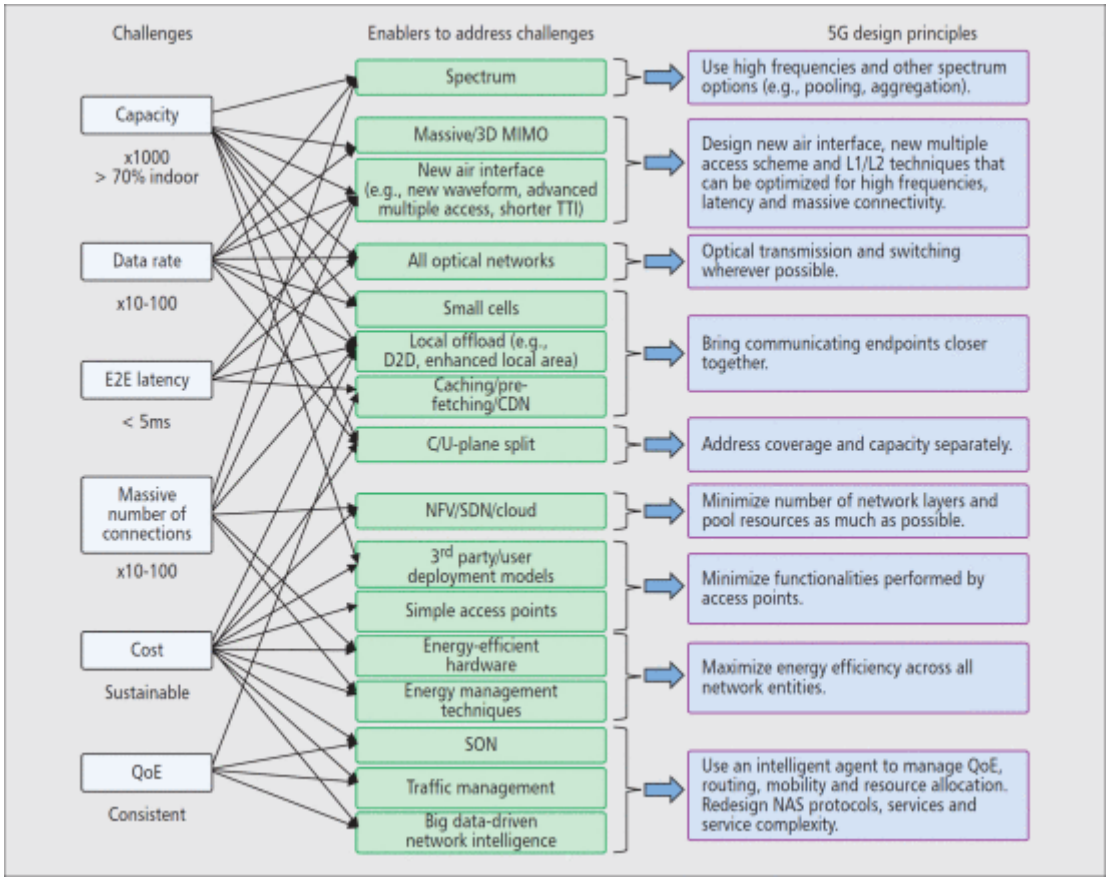


Figure 2.1.1.1. 5G challenge, facilitators, and design fundamentals [21]

The 5G system itself consists of three main components namely the user equipment (UE), the core network (CN) and the radio access network (RAN)

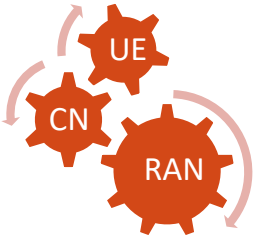


Figure 2.1.1.2. 5G System Components

2.1.2 5G Network slicing

The International Telecommunication Union (ITU) wanted 5G to be able to do more things than the traditional networks. They wanted 5G networks to be able to accommodate a lot more usage scenarios

than just mobile broadband. It is not an one size fits all so ITU defined three main usage scenarios which are:

- Enhanced Mobile Broadband (eMBB) which supports high peak data rates and high area capacity, spectral efficiency, extended coverage for activities that the majority of people do every day such as video streaming, browsing the internet, making voice over internet calls and other data consuming services. Offering a uplink of about 10 gigabytes per second and a download link of about 20 gigabytes per second.
- Ultra-Reliable Low Latency Communication (URLLC) which supports extra high reliability, strong security, extreme user mobility and ultra-low latency for applications such as smart vehicles which require low latency and high reliability. The latency in Radio Access Network (RAN) got 5G networks can operate at less than one milliseconds.
- Massive Internet of Things (MIoT) which supports low data rates, ultra-low energy, extra low complexity, and high connection density for use in applications that require low data rates and that send data less frequently such as machine to machine devices.

3GPP also adopted these three use case scenarios as the standardised network slices and then added a fourth one called vehicle to everything type communication (V2X) in Release 16 of the 3GPP specification.

A network slice configured with a standard setup for a specific vertical would be insufficient to meet the performance demands required to manage the vertical's operations and systems efficiently. Current networks, which follow a conventional 'one-size-fits-all' model, lack the flexibility and scalability needed to accommodate the varying demands for performance, security, availability, and cost. The existing network communication or a standard 5G network slice to cater for all these various requirements and services such as smart home, smart city, smart hospital, private networks with varying needs no longer efficiently support the various verticals. It is not cost efficient and optimised to provide the type of high performance required by the various applications. To solve this and achieve expected network services efficiently, a physical network is sliced into multiple virtual networks where resources could be dedicated to certain services or shared amongst applications. It is essential to break down the large, monolithic network functions, which are traditionally tied to various legacy hardware, into smaller, software-based modular network functions with varying levels of detail. These cloud-native functions can then be dynamically combined to create different network slices, each tailored to meet the specific requirements of 5G. By isolating network resources, it becomes easier to manage and optimize these slices to meet the specific demands of applications and services. This approach enhances the overall end-to-end Quality of Service. Each network slice comprises segments of the radio access network, core network, transport network, lifecycle management, and services. See Figure 2.1.2.1 below for the network slice components diagram.

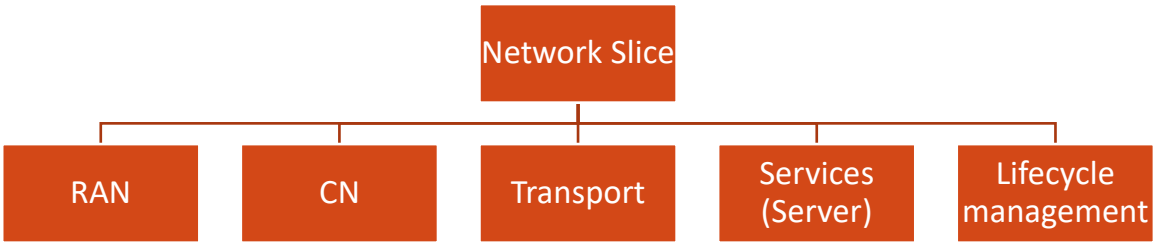


Figure 2.1.2.1: Network Slice Components

The mobile network is divided into two main categories which are the radio access network and the core network. Network slicing requires a slice of each layer. For example, to slice up an enhanced mobile broadband network, the RAN layer needs to be sliced, the transport layer needs to be sliced and the core network need to be sliced as well. This helps to provide the network function with the required characteristics. Network slicing helps to deliver various services in 5G network. It ultimately provides an independent management function of mobile broadband slice that helps to determine how many instances of the network function is required by the slice and to be able to dynamically adjust that based on traffic needs. Being able to slice up the core and provide various network functions based on the application requirements help to improve performance and bandwidth allocation for the various services that require this as they require them. Figure 2.1.2.2 shows the various layers sliced within the 5G network.

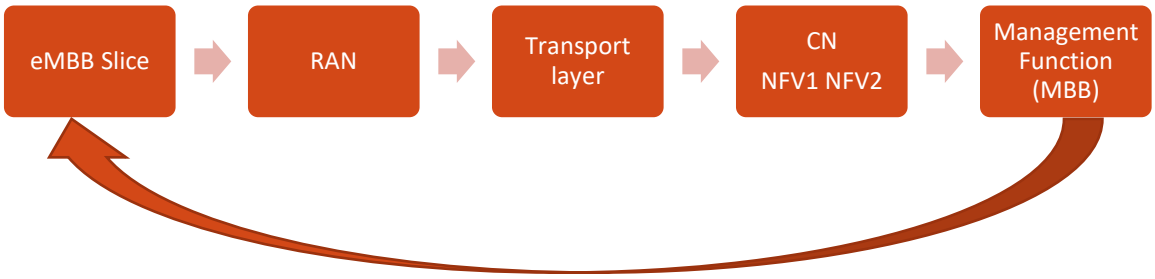


Figure 2.1.2.2: Layers of 5G network end to end slice.

5G enables network slicing with

- The spectrum it opened. There are wide range of spectrum available in 5G to support new services that will benefit from these spectra in the slice.
- The user subscription data in the Unified Data Management (UDM) is modified to incorporate network slicing information. A specific Data Network Name (DNN) is assigned to a specific network slice which is predefined in the subscriber profile.
- Software defined network which allows the separation of the user plane and control plane function and ultimately makes these scalable, flexible, and efficiently utilised.
- Network function virtualization allows hardware and software to be decoupled which ultimately also helps satisfy the 5G flexibility and programmability requirements.
- Next Generation Core which allows network functions to be allocated as required and could be dedicated to a slice or shared between multiple slices.

The network slicing is managed in the subscription profile of the user. The user's subscription profile contains, slicing information such as the type of slice for example URLLC, the number of data

network names for example internet and virtual private network. Aside the standardised slices defined by ITU and 3GPP, Mobile service operators can also define their own standardised slices which is generally referred to as non- standardised slices. This could be a VPN for example, or a service package name such as Platinum Internet, Bronze internet, or for vertical applications such as smart streetlights. The mobile operator also defines the slice differentiator for these. The transport layer can also be split into different slices. This can be achieved by dynamically assigning different routes to the slice or through segment routing, which divides the transport into smaller segments. These segments can then be assigned to specific network slices.

The personalised customer experience of meeting the quality-of-service requirements is what network slicing also aims to provide, a customised service based on network request to improve users' experience. Network slicing in the fifth-generation mobile network is becoming a very promising technology which is based upon multiple virtual networks being placed upon fully defined network functions and represents an end-to-end characteristic of a defined network. The need and demand for optimising these network slices is rising by the day as the prediction for more devices to be connected via internet and more complex Internet of things (IoT) applications are to increase due to the availability of 5G technology and beyond.

2.2 Related Work

this ChapterIn this Chapter, this thesis will review the existing studies on 5G technology in relation to meeting these requirements and demands required for efficiently orchestrating the subslices of a network slice. The research further delves into the review of existing study on network slicing in relation to 5G with its current challenges of optimising the orchestration of network slice resources. This review then led to review of the studies on Network function virtualisation and software defined network studies in relation to efficiently slicing network slicing and then ultimately reviewing the studies on machine learning studies on 5G network slice network orchestration as it relates to this.

2.2.1 Improving 5G Quality of Service

5G promises to be better than previous and current mobile network communications by offering network that is more flexible, with lower latency and better quality of service [4]. To achieve these specifications of mobile network with the proposed improved performance, J. Heinonen et al. [2016] proposes that the network function be deployed on the edge to reduce the latency and the mobility management which includes the gateway algorithm selection and the ability for handover to switch user planes seamlessly be improved to be able to meet the flexibility requirements. Whilst this helps towards meeting the 5G network flexibility requirements, it still requires more work to meet the flexibility in network resource demands required by various multiple applications running in parallel in need of network resources.

The study in [2018] also shows that 5G network will meet the requirements for scalability, configurability, and flexibility by leveraging on key technologies such as network slicing, software defined function, Network Functions Virtualisation, and mobile edge computing. The digital twin to simulate the 5G network sub slicing characterisation in the proposed revolutionary smart company will be discussed towards achieving the programmability and flexibility requirements 5G promises. 5G paired with Artificial intelligence hopes to change the world and provide opportunities to create new unique services especially with the existing machine to machine communications, Machine to human communications and peer to peer communications[17].

2.2.2 Resource Management in Network Slicing

Efficiently allocating the limited available network resource is crucial in efficiently orchestrating a 5G network slice to meet specific user device service requirements. N. Nikaein et al. [2015] defined network slicing as a complete network made up of configured network functions, cloud infrastructure and network application. It defines it as a network system which meets a specific use case requirement. A dedicated network for a particular service could be a slice of the network. This could be a private network or even part of the public network. The resources available on the cloud infrastructure and bandwidth available could be limited depending on the type of applications requiring the network and the characteristics of the network service required. The study explores network slicing in the form of a network function virtualisation ready network store in use with software defined network to provide slices on demand, however, the study did not investigate how to efficiently orchestrate this to achieve the concept. Caching of resource sharing was one feature that was studied by Qingmin Jia et al. [2019] in network slicing with the aim of improving its performance and providing customised network service as required. Caching will potentially reduce latency and improve the time to orchestrate a slice based on cached characteristics. While this study helps improves the end-to-end network slice user experience, it does not help with distribution of available slices or the orchestration of network slices by itself. The study by Qingmin Jia et al. [2] here will be used in this research as a deployment component in the CNaaS system design architecture.

Studies carried out by D. López-Pérez et al. [2009] shows that there is an increasing rise in the number of bandwidth hungry applications demanding for network resources and with the introduction of 5G, this is expected to double. To manage this limited available network resource for network slice, H. Zhang et al. [2014] studied management of interference to optimise network resource allocation. The study used the optimisation of subchannel allocation and power to achieve the proposed QoS requirements. Although the study is not 5G specific, the concept contributes to improved quality of service with reduced latency in 5G. However, the study did not consider the optimised orchestration of the subslices end to end in the study.

The end-to-end slicing of network was studied by Qian Clara Li et al [2016] and researched into vertical and horizontal network slicing. According to the study, vertical slicing is the sharing of physical resources such as core network, physical storage, and radio access network while horizontal slicing is the second phase or the build on the vertical slicing to enable mobile network slicing and improve user experience. The study discusses the sharing of the computational and communication resources across layers. It proposed an efficient way to slice the core network, radio access network and storage but did not discuss the orchestration of these resources or how to efficiently manage them. It wasn't compared against existing studies to verify the efficiency of the discussed method for end-to-end slicing.

Some of the challenges identified in the study carried out by H. Zhang et al. [2017] on the management of network slicing and optimization with the creation, deletion, maintaining quality of service is still a hard nut to crack. There is also the potential of the complexity of managing the available network resources to increase with increase in applications in the market that requires 5G connectivity. The study however presents network slicing management by slicing the network based on three key characteristics; Ultra Reliable and Low Latency Communication (uRLLC), Enhanced Mobile Broadband (eMBB), and /internet of Things (IoT) slice. The required minimum data rate, the required maximum transmits power of small cells, and the required interference threshold were all used in slicing the network into the three identified fundamental characteristics of the 5G network. Network was sliced

into the three key characteristics, but sub slicing was not considered in the research for enhanced and optimised resource allocation.

2.2.3 Network sub slicing

The resources available to the network slice for the emerging applications based on the new 5G network are limited and need to be managed efficiently and proactively for optimised performance. In the studies carried out by Singh SK et al. [16], network slicing was identified as a fundamental technology required to manage load balancing challenges that occurs in the multi-tenant 5G network. The study noted that various applications especially IoT applications require dynamic needs such as high spectral efficiency, low latency, and high reliability to meet the needs of these various IoT applications. Furthermore, a framework was proposed for sub slicing the logical network slice using the support vector machine (SVM) algorithm of machine learning. The study subsliced the logical network slice into three key categories, Platform based, Application based, and Infrastructure based. The study used machine learning to implement feature selection into these services and used K-means to cluster them based on the four key categories that was considered for identifying the service requirement for the network slice. The four key considerations are latency, power consumption, heterogeneity, and load balancing. The study [16] shows that with network sub slicing, bandwidth consumption in a network slice was improved and the available spectrum was efficiently utilised for the limited available network resources. While the study improved the network slicing allocation by sub slicing the network slice, the study did not consider other feature classifications to subslice and improve the performance of the subsliced network. It also did not base the selection of its machine learning algorithm on multiple machine learning algorithm models analysis but on known reputation of certain machine models for certain type of problems. However, not one size fits all as data varies from different verticals. This current research will also consider other key considerations such as required resource KPI, Central Processing Unit (CPU) usage, memory utilisation, disk utilisation with machine learning to orchestrate the subsliced network efficiently. Machine learning algorithm will be implemented to further subslice the slices and classify them into a more streamlined clusters for optimised orchestration of network resource. This will facilitate the virtualising of an organisation, or a revolutionised smart company as proposed in this study, efficiently orchestrating the network slice to fulfil the statement in [17] that suggested that 5G networks aim to allow the rise of new players, services, and business models in the market. The flexibility and programmability to achieve this has been made possible with the network function virtualisation technology and software defined network technology.

S. Ravindran et al. studies show that optimised resource allocation can be achieved by logically partitioning the network slices into network subslices [35]. The study proposed an algorithm for isolating network slices and improving throughput of the subslices. They showed the benefit of partitioning of network slices to subslices to increase service throughput, isolation and optimised allocation of resource based on varying user profile requirements. The study however does not investigate how network slices are partitioned into subslices or how subslices can be characterised for optimised resource allocation.

S. Ravindran et al. studies in [36] shows that slicing allocated network slice into subslices will allow the user equipment support multiple applications with the multi subslices. The study investigates optimising energy consumed during network sub slicing but does not investigate the KPIs that characterises the network subslices which could either serve application or vertical efficiently.

Another study by S. Ravindran et al. grouped users required network slices by service level agreements into network subslices [37]. The study further proposed a framework for optimising the delay and throughput by isolating the subslice resources across varying profiles. However, the study did not study the details of the service level agreements features used to group slices into subslices to satisfy the need of varying user profiles. There is no known literature that studies the further grouping of a subsliced network or the detailed KPIs of the network subslice used for classification. This current research will study the key KPIs to be selected and used for grouping the network subslices further into subsliced network end to end for optimised orchestration. Machine learning algorithm will be used for the feature classification.

Network slice management was studied in relation to radio resource management [48] with four methods identified to split radio access networks. The study uses user specific policy for the network slice admission control. These enhances the customisation of slice however the study does not investigate how this can be efficiently orchestrated. The four approaches examined addressed RAN slicing at the following four levels: spectrum planning, ICIC, packet scheduling, and admission control. The study carried out by Bin Han et al in [49] for optimum network slicing focuses on reducing capital expenditure for tenants while maximising revenue for mobile operators. This was achieved by proposing an iterative algorithm. Policy based admission control with the slice admission control was used to model the revenue for mobile networks operators. The study focused on revenue optimisation and does not investigate efficiently orchestrating and managing network resources for specific application services.

Efficiently sharing of available network resources especially among inelastic users who may change requirements in response to other tenants was studied in [50]. The research proposed a network slicing framework which consist of three main parts namely: user dropping, resource allocation and admission control. A game where each slice reacts and is dynamically orchestrated based on the reaction of another slice is proposed in this study. Auction based admission policy was used in this study. This research does not identify how the network slice features or grouped or efficiently orchestrated and classified.

Queue based slice admission control was studied to improve network slice quality of service and user experience [51]. Network slicing orchestration challenge was approached in this study by analysing users' behaviour and response to congestion. This study does not cover the network slice or efficiently orchestrating network slices based on application services need. Transparency strategy on the slice admission control policy in the infrastructure service layer to the virtual network operators was also studied to improve user experience.

Network properties were studied and analysed in the study carried out by Dapeng Wu et al in [52] to utilise network resources efficiently and profitably. A wireless network allocation framework was proposed in the study. Classification of features was done based on user preferences and grouped accordingly. Using user group concept, the study proposes social sharing of network slice resource based on user groups with same preferences or network resource requirement. This study does not use real data for its study and does not cover the optimised orchestration of features based on the network slice characteristics.

Stochastic geometry theory was used to propose an analytical framework to model a RAN for efficiently sliced network [53]. A non-sliced network was also studied and compare against sliced network in this work. Bandwidth and power were efficiently allocated by proposing a strategy of

efficiently allocating radio resources. Although this study shows improved spectral efficiency with orchestrated network slice, it does not cover the optimised utilisation of slices by applications or the optimised orchestration to meet heterogeneous application service's needs.

Abidi M.H. et al [71] studied the optimisation algorithms to efficiently slice 5G network slice. The study proposed the hybrid use of Differential Hunting Optimization Algorithm (DHOA) and Glowworm Swarm Optimization (GSO) algorithm which was tagged as GS-DHOA model. It further used deep belief technique as a learning technique for the model. The results indicate high performance compared to standard algorithms such as SVM however the study is based on use of only historical data for prediction and does not cater for real time 5G network slice requirement.

Different machine learning classifiers algorithms for adaptive and predictive management of 5G network slices was studied by Malkoc et al. [72]. In their study, six different machine learning algorithms were used manually in the study.

The use of machine learning algorithms to dynamically optimise 5G network slice was also studied by Martins, Joberto SB, et al [73]. The research further leverages Kubernetes technology to efficiently orchestrate 5G network slice resources. Slicing Future Internet Infrastructures (SFI2) architecture was proposed in their study. The SFI2 architecture incorporates machine learning algorithm in every stage of a 5G network slice life cycle to optimise the resource allocation and improve QoS. SFI2 treats energy as a scarce resource in their study and uses sleep mode to save energy when not in use and using renewable energy to prioritise providers energy resources thereby efficiently managing 5G network slice resources. The study strongly focused on secure and energy efficient network slicing.

Solving some of the challenges introduced by sharing virtual network functions on physical resources like virtual network embedding has also been explored in recent studies to efficiently manage 5G network resource allocation [74]. Based on the results shown from recent studies that Deep Reinforcement Learning (DRL) emerges as a solution for VNE, A. Bouroudi, et al [73] leveraged on this and proposed an Algorithm Selection (AS) paradigm by selecting the most suitable DRL agents from a portfolio of algorithms that have been used in the past to enhance the VNE solution. Their study adapted the Epochal Stochastic Bandit Algorithm Selection (ESBAS) which is an offline algorithm selection method to enhance the overall learning efficiency.

N. Shukla, et al [75] focused on the study of three machine learning algorithm towards optimising 5G network slice resource allocations. K-nearest neighbors (KNN), support vector machines (SVM), and linear regression were studied to predict 5G network slice for unknown device with the use of key performance indicators (KPI).

2.2.4 Network Function Virtualisation (NFV) & Software Defined Network (SDN)

With the rapid increase in mobile applications and need for improved user experience with rising demands of data rate, X. Xu et al. [3] studied the separation of user plane and control plane to improve the user experience by virtualising some physical network resources and implementing Software Defined Network (SDN). The traditional base station was also separated to improve the 5G capacity into Antenna Element (AE) and Central Processing Entity (CPE). The study also optimised the Virtualized Radio Resource Management (VRRM) by utilising resource pool to offer improved quality of service to users. The complexity of network slice resource allocation to meet diverse QoS requirements was not studied.

The study by J. Heinonenet al. [4] identified Software Defined Network (SDN) and Network Function Virtualisation (NFV) as key enablers to meet the flexibility requirement of the 5G technology to customise each network slice according to user's need. Furthermore, the study proposed that application and network functions be placed as close to the mobile network edge as possible to achieve low latency. The study shows that putting a low latency 5G network slice that consisted of Enhanced Packet Core (EPC) functionalities in the cloud edge and taking advantage of network function virtualisation with software defined network control, the quality of service was improved however the study did not show how various types of network slices reallocation could be optimised. This is particularly useful to fulfil the stricter requirement of 5G stated in [5] by 3GPP specification to reduce the latency to 10ms round trip in 5G systems for delay time between User Equipment (UE) and gateway compared to the 3GPP specification of 50ms -300ms for latency between UE and gateway depending on the Quality-of-Service requirement [6].Software defined network and VNFs also helps improve QoS and user experience with network slice in 5G by not only enabling network location to be easily configured or as an enabler for network slice orchestration but also by making it possible to decouple network process and separate network functions concerned. The study in [8] shows that with 5G, user plane and control plane could be separated in the Enhanced Packet Core (EPC) functionalities. Figure 2.2.4.1 shows the separation of the control and user plane in Virtual EPC compared to the standard setup of the Enhanced Packet Core.

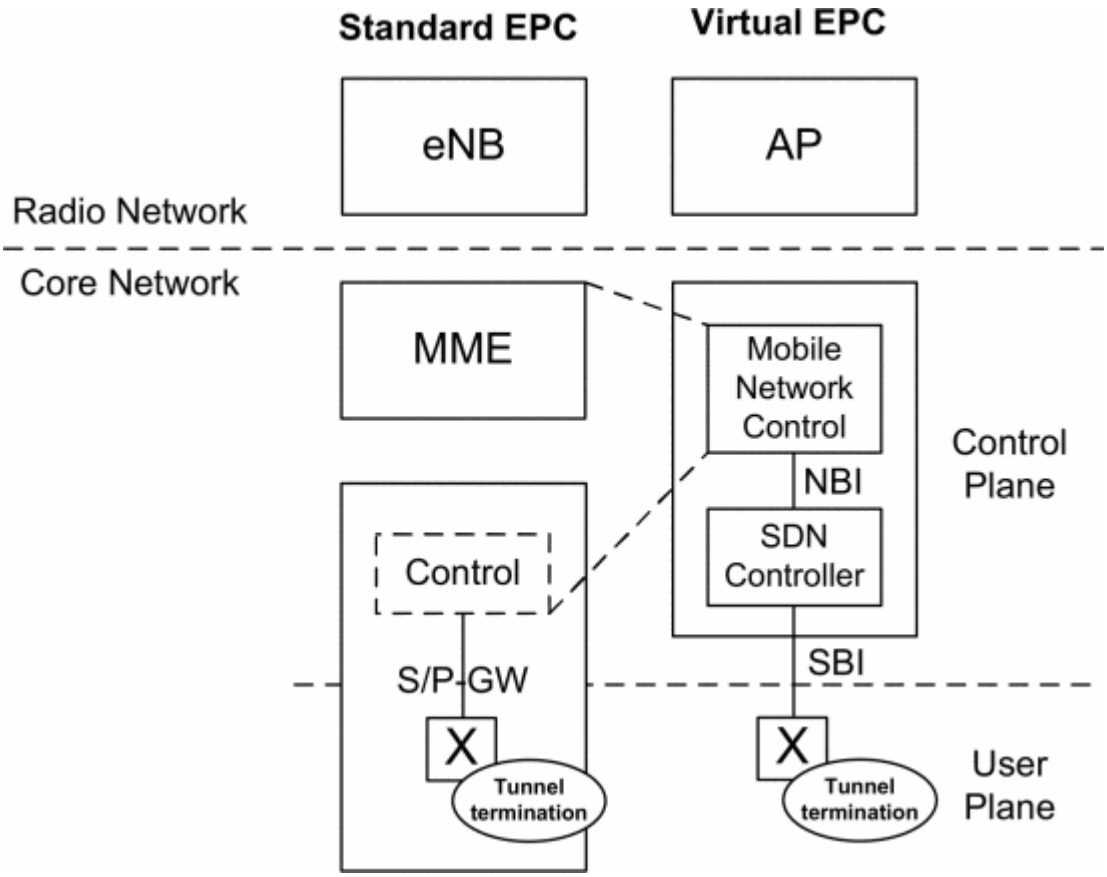


Figure. 2.2.4.1. Mapping EPC functionalities to VNFs [8].

A brief description of the abbreviated elements in Figure 2.2.4.1 is given below:

eNB: Evolved Node B – Base station for 4G LTE

AP: Access Point – Creates a WLAN.

MME: Mobility Management Entity – Manages user equipment mobility and access network.

S/P GW: Serving / Packet Data Network Gateway – PGW provides connectivity between user equipment and external IP networks. SGW routes incoming and outgoing packet data.

NBI: Northbound Interface is the interface to the top layer of a component that is responsible for output.

SBI: South Bound Interface: Enables the connection between the controller and the network device. The top layer of SBI links to the lower-level layer of the NBI.

The study by V. Yazıcı et al. [12] proposed the usage of an all software defined network architecture to allow the kind of flexibility and performance 5G technology promises. This study advocates for decoupling of user plane and control plane to be independently scalable, flexible, and configurable. While the study advocates for a programmable all Software Defined Network architecture, it did not study the optimised allocation of the network resources configured with the software defined network applications.

Although the study by R. Pries et al. [32] did not study either the slicing of the radio access network, transport or location awareness, the study however shows the possibility and ease of using software defined networks based on Nokia bell labs as well as Nokia's Cloud Application Manager to orchestrate a network slice dynamically.

Hernández-Chulde C and the other authors [22] identified Machine learning, Network function virtualisation and software defined networks in 5G as key technology which is key to network management and control to improve user experience and fulfil the requirements of applications/services for the fifth-generation networks. These identified technologies are the foundation of this current research study.

2.2.5 Machine Learning

Data transferred and needed for high data rate network capability is on the rise as more devices get connected and more innovative applications are deployed. Machine learning helps with analysing these huge available historical data as well as real time data to predict and provide information for allocation of resources efficiently [22]. The capability to extract data from the network, learn, forecast, and provide input to control and efficiently manage network resources is offered by Machine learning [22]. Network orchestration using machine learning will also help reduce cost drastically especially in an environment where the service deployed is pay as you go whereby users only pay for used or allocated network resource or slice. There are three main techniques that could help with network orchestration depending on the complexity of the data analysed which are supervised learning, unsupervised learning, and reinforcement learning. These techniques enable network slicing automation. Network slicing automation is essential due to large volume of complex network data required to be analysed within a short period of time to ensure low latency, availability, and optimised performance.

Studies have been carried out with the use of machine learning in various network management functions such as network resource management, service orchestration and creation, network

orchestration, anomaly detection, on-demand adjustment of policy, security, improvement of user experience and mobility management [23]. Machine learning and its feature selection capabilities are the primary foundation of this current research study. Figure 2.2.5.1 below shows the diagram of the three main machine learning techniques and some examples of their algorithms.

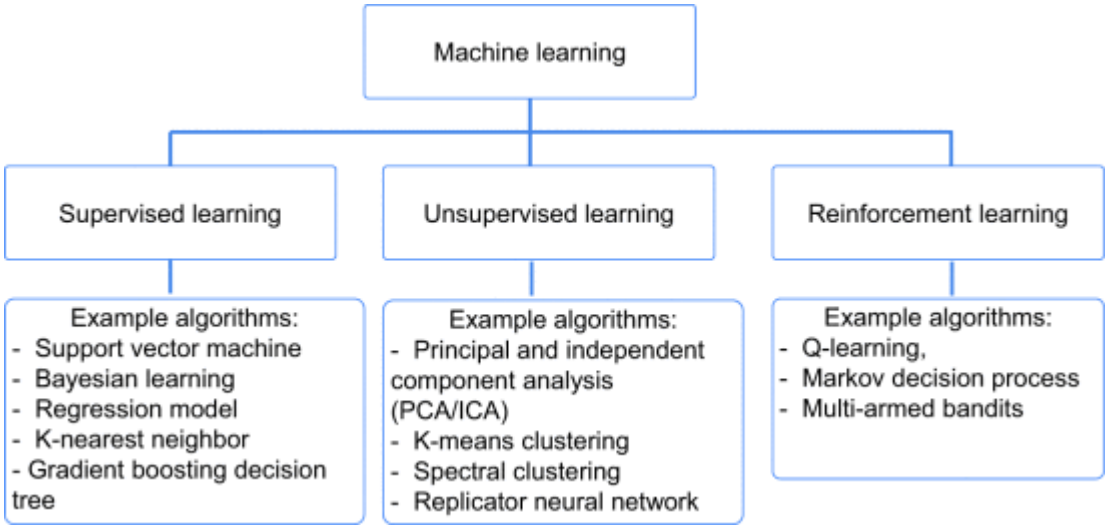


Figure 2.2.5.1. Classification of machine learning techniques [23]

A. Thantharate et al. implemented Random Forest as the supervised learning algorithm in their study for efficient and reliable 5G network slicing [25]. Supervised learning trains a model by supplying the system with labelled input data which are training data and classified by the algorithm into expected results or output [24]. Given a dataset D, the core objective of supervised learning technique is to map a given set size of data S with a given feature set F to produce a prediction function. The following objects illustrates the classification process on a given dataset:

- Set size → S = {S1, S2, …, S |S|}, where S is the size of the set S or the number of attributes in the set.
- Class or Feature label→ F: Target attribute; F = {F1, F2, …, F|F|}, where F denotes the number of features or classes and F ≥ 2.

Figure 2.2.5.2 shows the supervised learning technique architecture.

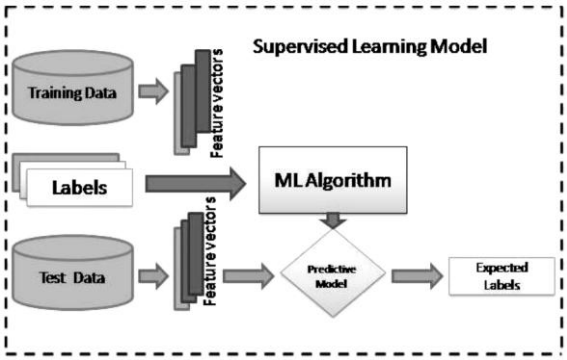


Figure 2.2.5.2. Supervised Machine Learning Model [24]

The study [25] used the rainforest algorithm to extract features from the dataset and classify them into the eMBB, uRLLC, mMTC and the main network slice. Rainforest was chosen over other types of supervised learning algorithm in the study because the dataset available was structured and huge. The performance of other algorithm against the dataset was however not analysed in the study. Rain forest performs well with training data on a large database and has high accuracy for predicting the missing data in any dataset correctly. The study used a combination of rain forest algorithm and deep learning neural networks to manage 5G network slice efficiently. Rain forest is however not a learning machine algorithm but used to split large dataset into tree structure.

Support Vector Machine and K-means were used in the study by Singh SK et al. [16] for feature selection to optically sliced 5G networks into the application based, platform based and infrastructure-based networks. Features such as IP address, port, size and number of the packets, network protocol type are used by the SVM algorithm coefficient in the study [16] to rank features according to their weight or frequency vector in the feature selection. For a given input dataset with various network features, such as IP address, port size, network protocol, and other network attributes, along with a structured output B.

- Let S = {S_1, S_2, …, S_k} be a set, where |S| denotes the size of the set, i.e., the number of attributes in the set..
- The set grouped based on the various features IP address and other network features. {a_i,b_i}
  - $A= [a_1, a_2,\ldots..a_i,\ldots..a_k]^T$,
  - $b = [b_1, b_2,\ldots..b_i,\ldots..b_k]^T$ with B = Ø.
- Sub features subset s= [1, 2, 3,…….n] with feature rank list B = Ø.
- Repeat sorting feature until s = Ø
- Strip training data to required indices: A0=A(:, s)
- Model classifier: $\mu=SVM-train\ (A_0, b)$
- Compute weight vector for feature selection ranking: w=∑kμkbkak
- Use criteria score value to calculi ranking.
  - G_k =W2k , Where, k=1,2, 3,…….|A|
  - G_k =Device Ranking with Criteria Score
- Compute other feature ranking criteria:
  - $F=arg_k min G_k$
- Use the union formal to update the feature set.
  - r=[s(f), r] /* s=features subset, r=rank, f=features
- Clean up with the smallest ranking criteria such that B= s(1:f−1, f+1, length(s))

An unsupervised learning technique trains a model by supplying unlabelled input data. Data are grouped into clusters and the agent learns the patterns in the dataset [23]. Unsupervised learning is also used for density estimation and dimensionality reduction. The K-Means algorithm which is one of the unsupervised machine learning algorithms was not only used by Singh SK et al. in their study but was also used by other researchers such as L. Le et al. [26]. It was used for data driven 5G network slicing. K-Means was chosen by the researchers because of its famous approach to clustering and scalability [26]. Other Machine learning algorithms which were surveyed in [26] for 5G network slicing based on data were Support Vector Machine (SVM) which is known for its patterning recognition ability, Neural network (NN), which is also used for patterning recognition as well as solving regression and classifications problems, Gradient Boosted Tree (GBT), which is also used for solving regression, and classification problems and Random Forest (RF).

Deep Belief Network is an unsupervised machine learning technique which was studied in [27] with Neural networks to efficiently slice 5G network. The neural network model total network was computed based on the hidden layer output. The equation below represents both the hidden layer and the output layer [27]:

$$\overline{E}^{(E)} = act\left(\widetilde{D}^{(E)}_{\widehat{B}w} + \sum_{q=1}^{In(t)} \widetilde{D}^{(E)}_{nh} GT_j\right)$$

$$\widehat{F}_s = act\left(\widetilde{D}^{(M)}_{\widehat{B}o} + \sum_{r=1}^{HIN} \widetilde{D}^{(F)}_{ho} \overline{E}^{(E)}\right)$$

Where:

$\widetilde{D}^{(E)}_{(\widehat{B}w)}$ = Hidden neuron bias weight

$\widetilde{D}^{(E)}_{(nh)}$ = Weight of the hidden neuron from the input neuron

In(t) = Input neurons

GT_j = Input features

$\widetilde{D}^{(M)}_{(\widehat{B}o)}$ = Output neuron weight

$\widetilde{D}^{(F)}_{(ho)}$ = Weight of the output neuron from the hidden neuron

HIN = Hidden inputs

The weight function is efficiently selected by computing the minimum error difference where

$E_r \text{and} \widehat{E}_r$ = the actual and predicted outcomes

$$ER = \underset{\{D^{NN}_{wf}\}}{\arg Min} \sum_{s=1}^{F(t)} \left|E_s - \widehat{E}_s\right|$$

A reinforcement learning technique trains a model by iterative learning through a series of trials through rewards and punishments [23]. The model learns and trains itself from experience. At each time t, The sensors receive percepts p_t which contains the rewards r_t from the control system, which is the environment and sends to the agent, the agent then sends the chosen action a_t from set of available actions back through the actuators. The environment or controlled system then moves to a new state s_{t+1} and the rewards r_{t+1} in this state are then calculated, then the percepts which contains the state of the transitioned environment $(s_{t+1}/s_t,a_t)$ is sent back again to the learning agent through the sensors. The model is successful with the many rewards it collects. The two key components of a reinforcement learning model are the policy that determines the reward allocation or the application of an action in a given state and the probability of transitioning from one state s to another s' with action a.

The probability $P(s'/s,a) = Pr\ (s_t+1=s'/s_t=s,\ a_t=a)$ [23]

Business models and vertical segments today are still being faced with the challenge of efficiently orchestrating network slices and are willing to pay extra despite the already expensive deployment to have a private slice which will enable them to provide value-added service to their clients[28].

2.3 Methodologies

Diverse techniques have been implemented by various studies to efficiently orchestrate 5G network slice.

Abidi M.H. et al [71] took advantage of two nature inspired optimisation algorithms, Deer Hunting Optimization Algorithm (DHOA) and Glowworm Swarm Optimization (GSO) by using the collective behaviour of a dataset to solve optimisation problems. It also leveraged on the power of deep learning by integrating Deep Belief Networks (DBN) with Neural Networks (NN), for optimised slicing g of the 5G network slice. The feature selection process was also enhanced in the study by optimising the weighted feature extraction. Although the result from the study shows improved 5G network slice orchestration, the methodology introduces complexity and increase in computation cost. The hybrid approach of DHOA and GSO coupled with DBN and NN may require a significant resource usage which will impact the practicality of this approach in a resource constrained environment. The use of deep learning techniques DBN and NN can lead to overfitting if the dataset available is not large enough for an efficient analysis and may have an effect in generalisation of unseen data or network slicing predictions. The methodology used was also based on simplified assumptions and didn't consider the practical user behaviours and dynamic 5G network conditions. The robustness and scalability of this approach in an environment with significant high load requires further investigation.

The use of Google Colab with python to compare the performance of multiple machine learning classifier algorithms was the approach employed by Malkoc et al. [72] in their study. The 5G dataset used in the study was obtained from Kaggle and the six machine learning algorithms that their performance were compared are: SVC, k-nearest neighbors, GaussianNB , random forests , decision trees and BernoulliNB.

To efficiently manage resources for 5G network slicing, a Kubernetes (FIBRE-IM) based domain interface manager (DOM-IM) was also used to manage the SFI2 deployment over a FIBRE-NG experimental network [73]. Resources such as containers, virtual machines (VMs), and bare-metal are allocated and deallocated within the FIBRE-NG islands. Multiple machine learning algorithms which included Decision Tree, Random Forest, K-Nearest Neighbors (KNN) and Multilayer Perceptron (MLP) were also evaluated and compared to find the best fit machine learning algorithm in each stage of a network slice life cycle. The research result showed MLP as the most efficient in their study with a longer training time and more computation resource utilisation compared to the other machine learning algorithms.

The algorithm selection (AS) paradigm used the ESBAS approach to select the most appropriate DRL algorithm from a portfolio of DRL agents based on past performances in an offline manner. A simulation test bed platform was developed to evaluate the AS approach. OMNeT++ is used to simulate the network topology. The components are then packaged into a Docker container and deployed to single machine. The metrics used to evaluate the performance of the AS approach are Acceptance Ratio and Selection Ratio.

A systematic approach was carried out by N. Shukla, et al in their study which was designed to use machine learning to optimise 5G network slice resource allocation. The 5q Quality Services' dataset used in the study was curated, analysed, and visualised to identify patterns and trends. The approach leveraged on MLR, KNN and SVM machine learning algorithms and focused on specific

parameters which are application type, latency and signal strength. The study found SVM and KNN to perform better than MLR.

A study [11] into management of network interference in 5G proposed sharing of spectrum rather than splitting the spectrum due to scarcity of the spectrum and the challenges faced in implementing spectrum to meet the rising demand. Sharing of a network slice by few low resource intensive applications will minimise the round-trip time to configure a network slice for each application and reduce processes taking place on the network. The study proposes a sophisticated algorithm for resource allocation where network is sliced seamlessly, and applications could be grouped together to share network slice to improve QoS and performance. Whilst this study implements the algorithm proposed in the study to group together low resource subsliced network, this study is further adding to this knowledge by proposing a cache which stores characteristics of previously requested network slice to quickly orchestrate the same configuration to prevent round trip and gain lower latency thereby improving user experience.

Studies in [12] also show that interworking of hierarchical controllers as opposed to a single controller helps with localising decisions efficiently within the network as well as providing optimised service provisioning locally to meet different QoS or grades of service required. This knowledge is implemented in further using machine learning to make use of local resources information demanded by applications to predict the configuration of network slice suitable for the application on the network. This study proposes a revolutionary industry 6.0 smart company as a case study with distinct and diverse requirements of QoS, SLA and KPIs across the vertical. This forms the basis of the proposed Company Network as A Service discussed in Chapter 6.

2.4 Challenges and Limitations

2.4.1 Limited Evaluation Metrics

Precision, accuracy, specificity and sensitivity were the primary performance metrics focused on by Abidi M.H. et al [71] in the study to evaluate the proposed GS-DHOA-NN+DBN approach. Other key metrics for evaluating the overall effectiveness of the model such as the robustness, the scalability and adaptability were not considered.

The study by Malkoc et al [72] also focused on precision, accuracy, F1 score and recall, metrics. These metrics are good with providing overall performance of the model however additional metrics such as confusion matrix, ROC-AUC or Matthews correlation coefficient (MCC) could offer deeper insight into how the model behaves particularly with dynamic datasets.

The evaluation metrics in N. Shukla, et al. study [75] focused primarily on Root Mean Squared Error (RMSE), Mean Squared Error (MSE) and R-squared. Extending the metrics might help provide a more comprehensive insight into the model's performance.

2.4.2 Assumptions and Simplifications in Methodologies

The study by Abidi M.H. et al [71] did not capture the complexity and dynamic behaviour of network environments as well as diverse user behaviours which may have an impact on the implementation in real-life deployment of their proposed methodology. The study also assumed there is availability of the required high volume of historical data to train and deploy the model. In reality, data fragmentation and privacy policy might impact this. A static and stable network condition was assumed during evaluation of this approach which is not typically the case in real-world 5G network environment.

A small dataset size was used in the study by Malkoc et al. [72] which shows good performance however the restriction to fixed dataset sample size could impact the ability of the model to adequately generalise and learn from the data.

DOM-IM and DOM-MON interfaces implemented [73] may not generally support all potential domains thereby requiring further research need on customisation and development.

Static number of VNF-FGs were used in the study by A. Bouroudi, et al. [74] which does not reflect the real-world scenario of dynamic departure and arrival of VNF-FGs

The model performance in N. Shukla, et al. study [75] may be specific to a certain use case or 5q Quality Services dataset. Further validation against other dataset to mimic the dynamic nature of 5G network environment may be necessary. Although SVN and KNN showed superior performance, this may not be valid for other 5G network scenarios as other environments might perform better with some other machine learning algorithm.

2.4.3 Homogeneous Network Slices

5G network slice requirements and characteristics can vary drastically from one user to another however Abidi M.H. et al [71] considered 5G network slice as homogenous in terms of characteristics and requirements. Some static parameters which could not be dynamically adjusted and configured to adapt to users varying needs and dynamic network conditions were used in the algorithm. Sample size such as the population size was fixed throughout the evaluation. Other fixed parameters such as the initial weight value in NN, learning rate in NN, threshold values for updating solutions in DHOA, fixed attraction and repulsion coefficient as well as other static parameters shows the inflexibility of this approach. Making the parameters dynamic and adaptive will improve its flexibility and ability to cater to varying network conditions as well as diverse users' requirements.

The 5G network environment is dynamic and models need to be adaptive and robust. The possibility of 5G network conditions changing rapidly and ability of models to adapt to this real-world 5G network environment characteristics was not addressed in the study by Malkoc et al. [72].

The SFI2 architecture may require different machine learning algorithm for different domain and there is need to continuously update the environment with the best fit machine learning algorithm[73]. The machine learning model proposed in their study may not adapt quickly or scale well to meet the demands of the expanded network thereby impacting performance.

The effectiveness of a DRL algorithm [74] can vary significantly with the dynamic nature of 5G network slice environment.

2.4.4 Scalability and Performance

As the network grows, maintaining the proposed SFI2 may become complex and challenging impacting the overall performance [73]. Central point's such as control plane and data plane managing critical functions may become a bottleneck as the network size and demand increases slowing down the overall system performance.

The limited available computational resource at the edge might pose an additional challenge for efficiently selecting DRL algorithm. The SFI2 selects an algorithm offline which does not capture the real-world scenario of 5G network environments which changes dynamically.

2.5 Future Directions

While Abidi M.H. et al [71] offers a promising approach for optimised slicing of 5G network slicing, future work may cover the advancement of the model to be flexible and cater for carrying network conditions as well as diverse user requirements. Future research could focus more on evaluation on similar real-world 5G network environments or taking the characteristics of real-world dynamic environment into consideration.

The comparisons of various machine learning algorithms help identify the best performing machine learning algorithm for the available dataset [72]. Future studies could look into making the model more flexible to explore a more advanced algorithm and add to the comparison in an automated fashion. Security and standardisation in 5G are crucial. Future studies could also focus on integrating security policies as well as regulatory policies in the model deployment.

The SFI2 [73] future study can address the scalability and interoperability challenge of multi domain network's technology and domain integration. Future work may also delve into optimising the machine learning algorithm selection model by improving the data collection and labelling process as well. The ability to reconfigure the network slice based on the dynamic user need and incorporating it into this promising SFI2 architecture will also be valuable.

Extending the research by A. Bouroudi, et al [74] to use an online selection approach will contribute towards the paradigm adapting to the dynamic 5G network environment.

Extending the studies by N. Shukla, et al. [75] to incorporate more data sources either collected live from a 5G network environment or synthetic in the future might help improve the scalability and robustness of the proposed model. Future work can also improve the data processing technique and implement advanced processing techniques. The future work can also involve real-time data update consideration.

2.6 Summary

From the analysis of the above studies. Orchestration of network slicing is still faced with challenges of efficiently utilising the available network resources and network subslices has been under studied so far. This current research project aims to further the existing study by sub slicing the subsliced network slices into more logical groups using machine learning to efficiently utilise the available network resources efficiently. This study ultimately proposes an optimised end-to-end network slice orchestrating framework which is intelligent and self-organises the network by dynamically learning unknown networks and sub slicing available resources to improve quality of experience as well as quality of service. The top highlighted existing studies have been summarised in the Table 2.3.1 below.

In this study, we are proposing an end-to-end network softwarisation of a vertical application for drastic cost reduction and optimised network orchestration. This can be referred to as Company Network As A Service (CNaaS). We present a virtualised Smart Industry 4.0 company that runs with the same concept as a digital twin. The study delves into real-time management of network resource allocation within the required latency specification time frame. The characteristics and categories for sub-slicing the network are identified and the machine learning algorithms to orchestrate the network slices seamlessly are then presented.

The rest of this study will be structured as follows: A brief overview of the study of 5G network slicing is firstly discussed, then the network functions based virtualised smart company with its requirements is described. Next, the network characteristics required for the various departments in the

organisation is discussed. The proposed architecture for a seamlessly optimised network subslice is then demonstrated followed by analysis and presentation of the machine learning algorithm to predict and forecast network slice requirements.

Table 2.3.1. Highlights of existing studies researched

Reference	Existing Studies	Observation	CNaaS Contribution
3, 4, 8, 12, 32	Proposed various models for admission control strategies by studying optimisation of the RAN slice. The concept of separating the control and user planes were core in the studies. SDN and NFV implementation also formed major part of the studies.	The studies showed optimisation of the Virtualized Radio Resource Management (VRRM) by utilising resource pool to offer improved quality of service to users. However, the complexity of network slice resource allocation to meet diverse QoS requirements was not studied.	A flexible and scalable framework with self-organising network capability called Company Network As A Service (CNaaS) was proposed in this study to dynamically and efficiently allocate network slice resource in a simple or complex network architecture by fulfilling the varying network resource needs of devices on the enterprise network.
16, 23, 25, 26, 27, 72, 73, 75	Machine learning algorithms were used to study the optimisation of network slice resource allocation. SVN, K-Means, DL neural networks, CNN, RF are AI and ML algorithms that have been used in various studies for optimising network slice resource allocation. Containerisation technology was also studied in some cases using Docker containers and Kubernetes for orchestration.	While some of the studies selected machine learning algorithms based on their simplicity and popularity without evaluating their performance against other algorithms, some few studies compared their selected algorithms against few selected algorithms.	This work is recommending RF based on research and comparison against twenty-nine other algorithms rather than based on popularity or data size. Although the proposed framework is open platform with flexible and dynamic use of technologies or ML algorithms, research was carried out to suggest a default ML algorithm in the proposed model for enterprise networks. A model with RF as default algorithm for classification and regression is proposed with flexible and dynamic framework to auto select best ML algorithm for given dataset.
1, 2, 9	Various methods which are not machine learning and AI based were also studied to improve user experience and QoS such as the concept of improving mobility management, caching, managing interference,	While these studies contribute to the concept of network slicing, and improves user experience of the end-to-end network slicing, they did not discuss the optimisation of network slice in an environment with diverse or varying	The concept of enhanced subslice (eSS) was introduced and suggested in this study to efficiently allocate network resources to the diverse applications on a network in need of varying types of network slices and manage the limited available network resources.

		requirement and did not study sub slicing to reduce waste and efficiently orchestrate the network slice	
7,49	Proposed models for allocating network resources with the goal of increasing profit and reducing capital expenditure.	The study showed how to minimise cost by prioritising requests to determine which to accept and deny or delay. It does not look into how to efficiently orchestrate network slice to meet the varying user devices needs.	This study leverages on the concept of self-organising networks (SON), SDN, NFV and machine learning to dynamically orchestrate network slice thereby only allocating what is required by an application. It automatically reuses the slice elsewhere if no longer needed or does not allocate there by making it cost efficient by only paying for what is used ad also managing the limited available resources.
35, 36, 37, 48, 50, 51, 52, 53	Studied and suggested the concept of network sub-slicing to meet the varying users request and optimise throughput by proposing various methods such as an isolation-based subslice, energy efficient grouping of slices, gaming concept for auction-based admission control, queue-based admission control, users personalised preference and slicing the RAN	Although the studies discuss the concept of sub slicing and grouping of user devices or applications to access the subslices, the concepts studied are based on pre-defined categories and pre-defined number of groups. End to end optimised orchestration of network slice were not studied.	This current work proposes an application model to analyse diverse network slice real time requirements, extract important features and KPIs, group them into cluster of varying network slices requirements, cache, monitor and continually learn to efficiently orchestrate the network slice from the dynamically formulated enhanced subslices This study also identified new classification features to subslice a slice further.
71,74	The hybrid use of different types of machine learning algorithms such as DHOA and GSO, DRL and ESBAS were studied to improve the scalability and robustness of efficiently allocating 5G network slice resources.	The study highlights innovative and significant advancement in the approaches taken to optimise the 5G network slice resource allocation. With the substantial progress made, the challenge of adapting the model to fit the dynamic nature of 5G network environments remained. In addition to using static parameters in the studies, a single data source was used which	This study is going to evaluate the proposed model against two data sources collected over 5G network environment and synthetic data sources to test against dynamic data and environment.

		could limit the performance to a specific use case and impact the result when deployed to real-world environment.	

3.0 Design and Architecture

3.1 About The Proposed Model, Framework, Streamlined VNO and Vertical

The design architected in this research seeks to address the existing challenges highlighted in Chapter 2 by developing an advanced intelligent end to end AI model for predictive 5G network slice management and optimised proactive network resource orchestration for dynamic network using machine learning techniques. The proposed intelligent novel framework simplifies network slice orchestration right from the collection of data or the detection of application on the network in real-time to the allocation of required slice to the diverse applications over the network. The designed framework is smart and automated with little or no human interaction need. This study proposes a framework and model that will be able to predict the resources required on a network from learning the networks using reinforcement learning, use the 5G self-organising network, implement service chaining and machine learning pipelines to automate the collection of network data profile, full management, continuous monitoring and processing of the data as well as self-configuration and self-optimisation automation in real time to meet the diverse types of services in a smart company network with heterogenous networks, homogenous network, microcell network or ultra-dense networks.

This study further proposes an intelligent, dynamic, and flexible vertical by applying this 5G network feature of catering to diverse vertical network slice requirements with the proposed model as a case study. In this research, the proposed vertical is referred to as a smart company. Different sectors in a typical organisation or company require different services, KPIs, SLAs requirements for their network needs. The smart company dynamic services and multiple applications require the standard network slices defined by 3GPP; eMBB, uRLLC, mIoT and V2X. There is need to efficiently allocate the network slices to each required application. The optimised orchestration of network slices to these diverse applications on the dynamic network is achieved by providing customized network services with the limited network resources available while reducing capital expenditure and operation expense. It is natural to slice a physical network into several logical virtual networks so that each one can be customized and dedicated to a unique type of use cases [39]. The proposed smart company has a mix of mobile application services, diverse sensors, smart applications, and devices requiring machine to machine as well as machine to human communications, processing intensive applications and services with varied network performance requirements. The smart company networks will support multi-tenant applications and multi-service network. According to the studies by A. Kaloxylos [38], network slicing in 5G network communication is a key enabler to meeting these varied operational and functional requirements with the desired flexibility. The study further defines new KPIs and framework to efficiently implement and orchestrate this 5G feature.

The proposed framework aims to revolutionise the way Virtual Network Operators (VNO), Mobile Network Operators (MNO), Infrastructure Providers and Application Service Providers (ASP) interact to orchestrate a network slice. It will streamline the process thereby improving quality of experience by reducing wait times, communication delays and reducing running costs due to newly automated layers and reduction of manual processing. Virtual network operators and mobile network operators can now directly use the framework to cluster the subslice, initiate network subslice, access cache, monitor resources, analyse live and historical network traffic data, service chain network function and ensure policy enforcement. This will potentially remove the need for a virtual network operator to wait on Application providers to pass on classified data and then later wait to send request to

infrastructure provider before being able to deploy the required network subslice. The virtual network operator can instead implement the self-organising ONSSO framework and possibly publish events to a message queue. Providers such as Application service providers, mobile network operators, and infrastructure providers can subscribe to events on the ONSSO message bus and act accordingly thereby improving quality of experience, reducing the risk of denial of service being able to forecast and plan physical network resource needs as well as increasing reliability. This study enables the optimised performance of an MNO and the seamless collaboration between the VNO, ASP and infrastructure provider to effectively send request to the Network Slice Selection Function (NSSF) which then utilises the Network Slice Template (NST), Policy Control Function (PCF) and Network Slice Selection Assistance Information (NSSAI) data to allocate the required resources by also delegating subnet selection tasks to Network Slice Subnet Selection Policy Function (NSSP). Figure 3.1.1 shows a summary of the Network slice orchestration cycle and the activities in the Application service provider layer, the Virtual Network Operator (VNO) layer and the infrastructure provider layer. The Before on the left-hand side represents the existing Mobile network operators process cycle and the proposed framework in this study while the Right-hand side represents the proposed streamlined process this framework enables thereby efficiently allowing the collaboration of mobile network provides and reducing latency during network slice orchestration. Online user equipment are all devices and applications on an enterprise network which require 5G network slices. These could beIoE, computers, smart devices or any application with diverse network slice requirements, The existing process of network slice orchestration requires the VNO to wait for the Application Providers to send full details of request and the VNO interacts with the infrastructure provider layer by the SDN and NFV as shown in Figure 3.1.2. It also shows the layers where the suggested network tools in this study for monitoring, collecting data and for training are applied. Currently, the collection of data, Its storage and classification are done by the ASP. These are then passed on to VNO by the ASP. The VNO clusters the subsliced network slice requirements, initiate NS, monitors, access cache for faster deployment and reconfigures. From here, the IP receives the required information from the VNO. And then monitors the slice orchestration, reconfigure and orchestrate NS. The VNO then service chains the NF for optimised resource allocation and deploys the required NS to the user devices. With the enhanced sub slicing model capabilities, the process will be streamlined giving the VNO the ability to smartly orchestrate slices to meet specific application needs on demand. The framework simplifies this and aims to reduce latency. Mobile network providers can access user device requirements in real time by subscribing to the message bus within the ONSSO framework. This improves their ability to predict enterprise network needs in real time and proactively allocate required resources thereby improving QoS.

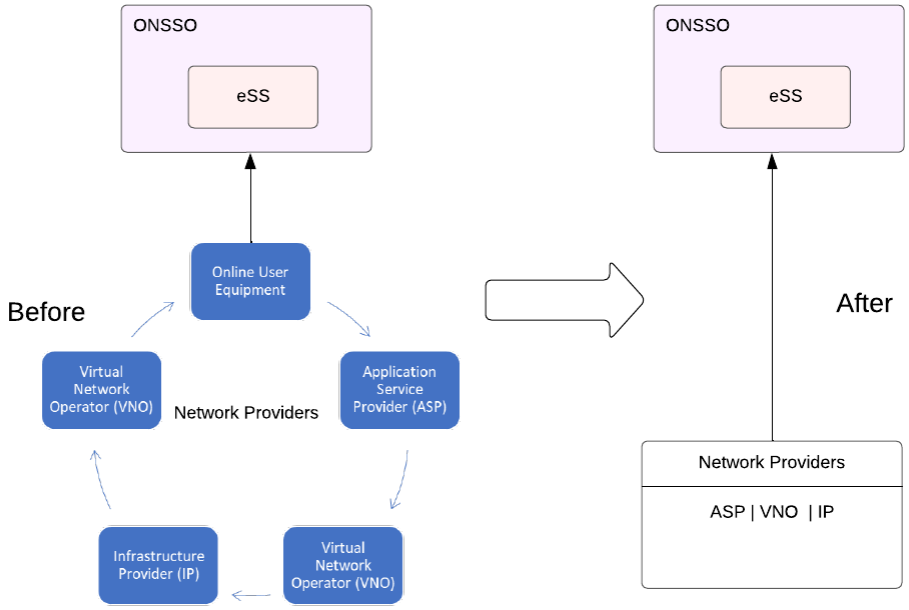


Figure 3.1.1: NS Orchestration Cycle From Application Layer to Infrastructure Layer

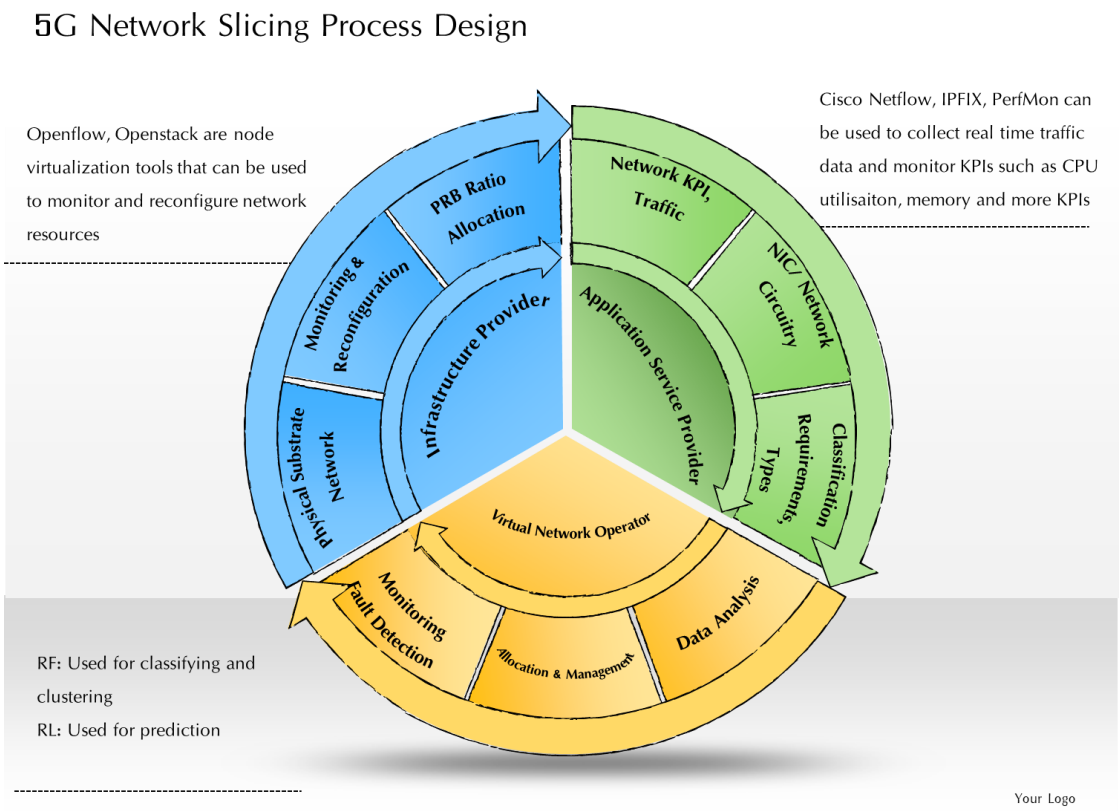


Figure 3.1.2: 5G Network Slicing Process Design

The methodology employed for developing the proposed artificial intelligent model pipeline in this study for efficiently orchestrating 5G network slice is shown in Figure 3.1.3 with the resource orchestration continually improved by generative AI and reinforcement learning. Generative AI is used in the proposed framework in this research to achieve three main goals. Firstly, to evaluate the dynamic network slice configuration based on the given network conditions and requirements. Secondly to evaluate the automation of the slice optimisation by reconfiguring network topology dynamically, automatically adjusting the slice to optimise the usage of the available network resource and scaling slice capacity to meet requirements. Thirdly, to evaluate various network scenarios with the proposed machine learning pipeline model. It generates synthetic data and implements diverse configurations. The implementation of the generative AI and the evaluation is detailed in Chapter 9 (Evaluation)

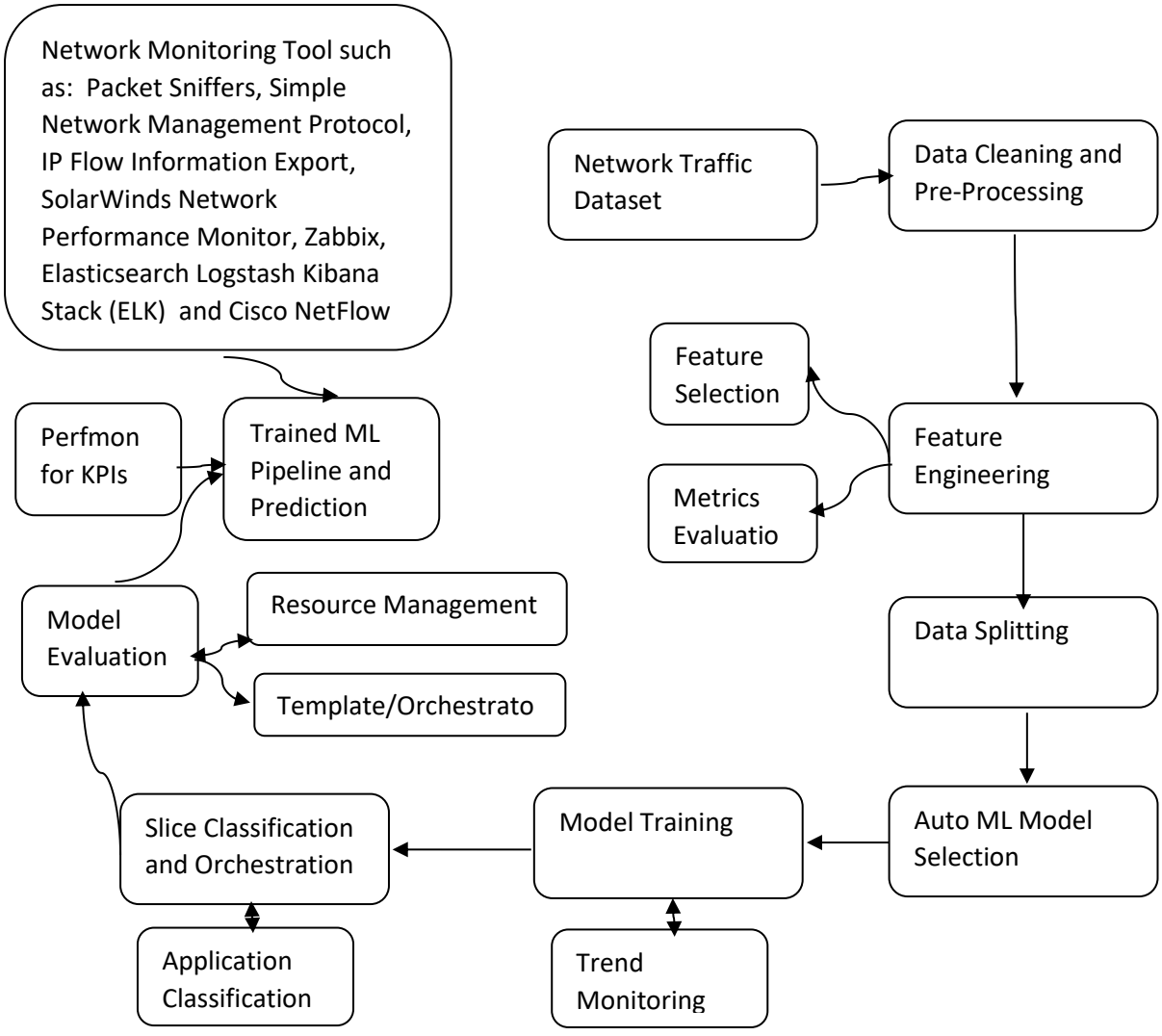


Figure 3.1.3: Network Subslice Orchestration Methodology

Figure 3.1.3 above shows that data that is analysed and modelled could be collected in real-time over the network or could be an historical dataset. These are then processed, with the models trained, applications classified, slice predicted, allocated and monitored. Applications with already defined

network template can also call this machine learning pipeline model, which then calls the orchestrator. If the available resource is not enough to meet the demand, the model suggests the best fit available resource and orchestrates that. The selection of classification models that best fit the dataset or the network being monitored, the generation of metrics for the network performance, the identification and grouping of applications based on their characteristics as well as the orchestration of the 5G network slices was automated as part of the machine learning pipeline model proposed. This Chapter will discuss the technology and strategy to efficiently orchestrate the slice at each stage. An evaluation of each step is provided where possible.

3.2 The 5G Quality Metrics

For an end-to-end adaptive 5G network slice orchestration framework, the 5G network slice requirements of all active user devices on an enterprise network need to be effectively derived. Without accurate performance metrics, it would be impossible to monitor and optimise network slices effectively. Raw traffic data obtained while monitoring user devices over a network typically includes packet-level details such as timestamps, source and destination IP addresses, packet sizes, and protocol information. They do not typically include metrics required to formulate the network slice templates such as latency, throughput, jitter, and bandwidth in a form ready for use. While this raw traffic data is useful for understanding the basic flow of traffic and provides the foundation for monitoring, it does not provide direct insights into critical network performance indicators like latency, bandwidth, or jitter. These indicators are crucial for monitoring the quality of service (QoS) and ensuring that 5G slices meet the specific needs of different applications. They must be derived or inferred from the raw data through calculations and analysis. Traffic data typically includes raw data like packet size, packet count, timestamps, source/destination addresses, and other header information, but calculated metrics like latency, throughput, and jitter need to be derived from this raw data. These metrics are continuously derived within the proposed ML-based eSS model as part of the inbuilt function for optimising slice orchestration. This enables the implementation of closed-loop automation, where orchestration systems can make real-time adjustments based on current network conditions. For example, if latency metrics indicate a delay, the system can automatically adjust the resources allocated to the slice to restore performance.

Tikhvinskiy et al [70] divided the 5G quality parameters into three levels. The three levels are Quality of Service (QoS), Quality of Experience (QoE) and Network Performance (NP). Throughput, Latency, Availability and Reliability were identified as the key QoE indicators. These quality 5G quality parameters according to Tikhvinskiy et al [70] were then used as the 5G quality metrics in this research. The dataset [67],[68] used in this research provides insight into the categorisation of the allocation of different slices to different types of user equipment. Evaluating these datasets also shows that the packet loss rate and packet delay budget could be seen as forming integral element of 5G Quality of Service Identifier (5QI) [70].

The defined metrics in this study are automatically calculated as a new application request is discovered on the network in real time. This was implemented as part of the proposed model pipeline. The following calculations were auto computed on the dataset to obtain the required subslice network functions required to orchestrate the slices for the network:

Duration was calculated in seconds by converting 'Flow.Duration' from microseconds.

$$\text{Duration_Sec} = \frac{\text{Flow. Duration}}{1000000.0}$$

Bandwidth in Bytes per second was evaluated from 'Flow.Bytes.s'

$$\text{Bandwidth} = \text{Flow.Bytes.s}$$

Latency in seconds by assigning the value of 'Duration_Sec'

$$\text{Latency} = \text{Duration_Sec}$$

Throughput was evaluated in Bytes per second.

$$\text{Throughput} = \frac{\text{Total. Length. of. Fwd. Packets}}{\text{Duration_Sec}}$$

Data rate was evaluated in Bits per second.

$$\text{DataRate} = \frac{\text{Total. Length. of. Fwd. Packets} + \text{Total. Length. of. Bwd. Packets}}{\text{Duration_Sec}} * 8$$

Jitter was evaluated in seconds

$$\text{Jitter} = \text{Flow.IAT.Std}$$

Packet Loss was evaluated as a percentage.

$$\text{Packet_Loss} = \frac{\text{Total. Fwd. Packets} - \text{Fwd. Packet. Length. Mean} * \text{Total. Fwd. Packets}}{\text{Total. Fwd. Packets}} * 100$$

Data Volume was evaluated in Bits.

$$\text{Data_Volume} = \text{Total.Length.of.Fwd.Packets} + \text{Total.Length.of.Bwd.Packets}$$

Speed was computed in Bits per second (bps)

$$\text{Speed} = \frac{\text{Data_Volume}}{\text{Duration_Sec}}$$

Packet Delay was evaluated in seconds.

$$\text{Packet_Delay} = \frac{\text{Duration_Sec}}{\text{Total. Fwd. Packets} + \text{Total. Backward. Packets}}$$

Reliability was evaluated in percentage.

$$\text{Reliability} = \frac{\text{Total. Fwd. Packets} + \text{Total. Backward. Packets} - \text{Packet_Loss}}{\text{Total. Fwd. Packets} + \text{Total. Backward. Packets}} * 100$$

To further evaluate this, the AutoGluon machine learning algorithm was run against the dataset as shown in Figure 3.3.2 and it eliminated useless columns thereby validating the meaningful columns to help enhance QoS. A snippet of the result showing identified useless columns and why the algorithm has predicted them as useless is as follows:

Useless data sent over the network that will not have much impact on the Quality of Service of the network were identified by the algorithm. The AutoGluon algorithm found the following useless because they carry no predictive signal: ['Bwd.PSH.Flags', 'Fwd.URG.Flags', 'Bwd.URG.Flags', 'CWE.Flag.Count', 'Fwd.Avg.Bytes.Bulk', 'Fwd.Avg.Packets.Bulk', 'Fwd.Avg.Bulk.Rate', 'Bwd.Avg.Bytes.Bulk', 'Bwd.Avg.Packets.Bulk', 'Bwd.Avg.Bulk.Rate', 'Label']. It also found irrelevance because these features have the same value for all rows: ['SYN.Flag.Count', 'ECE.Flag.Count', 'Fwd.Header.Length.1', 'Subflow.Fwd.Packets', 'Subflow.Fwd.Bytes', 'Subflow.Bwd.Packets', 'Subflow.Bwd.Bytes']. These features were not used to generate any of the output features. Features can also be unused if they carry very little information, such as being categorical but having almost entirely unique values or being duplicates of other features. They don't need to be present at inference time. These are: ['Fwd.Header.Length.1', 'Subflow.Fwd.Packets', 'Subflow.Fwd.Bytes', 'Subflow.Bwd.Packets', 'Subflow.Bwd.Bytes', 'SYN.Flag.Count', 'ECE.Flag.Count'].

3.3 The Dataset Flow Design

One of the main goals of this study is to design a scalable and intelligent framework that will smartly auto orchestrate network slices with the required resource on demand. The goal is to orchestrate an optimized network slice that maximizes throughput, minimizes latency, ensures sufficient bandwidth, and meets other network slice requirements for the diverse user devices on the network.

As shown in Figure 3.1.1, network data is required either in real time or historical data to learn the network and efficiently propose a model to orchestrate 5G network slices to efficiently meet its needs.

In order to simulate diverse network traffic data and evaluate the performance of the proposed model and framework with diverse network data, three types of datasets were evaluated which are real-time datasets collected which is measuring the data packet flow over a network [68], historical 5G datasets which contain the key performance indicators of the devices network available on the observed 5G network [67], and generative AI for synthetic data simulation of various configurations.

Real-Time Data: Similar real time dataset collected from online applications will be used as input to the model and to retrain deployed models. Simple Network Management Protocol (SNMP) [51], IP Flow Information Export (IPFIX) [52] ,Cisco NetFlow (CN) [53], Open source monitoring software such as Zabbix, Grafana, Nagios and Prometheus, Packet sniffers such as tcpdump and Wireshark, as well as Elastic Logstash Kiana (ELK) stack are examples of some network monitoring tools that can be used to collect real time network data for input. PerfMon can also be used to monitor key application KPIs such as memory reserved, disk, engine, GPU and CPU utilisation. Both network packet data and computer performance monitor data are used to determine the application network slice characteristics and requirements.

Other 5G Dataset: The first dataset [68] is a similar organisation network dataset which was collected at the Universidad del Cauca for multiple types of application services to classify applications originally. This offline dataset was used to train and test the model and was used as part of the method for developing the proposed framework in this study. The dataset collected from Universidad del Cauca network used in this research consist of 87 network features collected within a month while monitoring network traffic and collecting data from source IP to destination IP [42]. The dataset consists of real-time data with over 3 million observations. It is significant, containing 87 measured features and representing a diverse range of data types, sources, and categories. The second dataset [67] analysed consist of the KPIs of device networks, the time of the day to compare data against as well as weather, slice type, packet loss rate, packet delay budget, technology supported and other features.

Generative AI: Generative AI was used to simulate the synthetic 5G network dataset. The historical dataset obtained from the real-life network for developing a machine learning model to efficiently orchestrate a 5G network slice was also split into 3 parts. This was divided to be able to use each part of the real-life data with varying conditions to validate the efficiency of the proposed trained machine learning pipeline model.

Figure 3.3.1 shows the high-level process flow of data from data collection to prediction. All user devices on the company network such as vehicles, mobile phones, IoT devices, IoE, and other smart devices that require network resources are detected and network traffic data collected via inbuilt network circuitry or network interface cards.

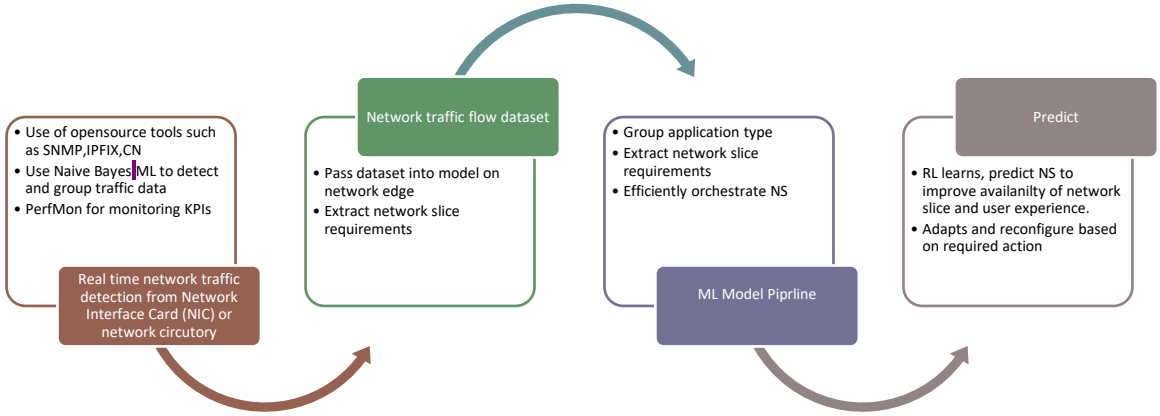


Figure 3.3.1: Network Slice Orchestration Flow from Data Collection

The proposed framework was tested on a similar dataset to validate and verify the framework. The network slice orchestrator architecture is based on the technical specification and requirements by 3GPP for network resource management and sharing as shown in Figure 3.3.2 [45]. This research uses the analysis of the network pattern analysed in this dataset with the offline key performance indicators of the application services characteristics collected to define and classify the network slices using machine learning clustering algorithms. Each orchestrated network slice instance is characterised based on the network characteristics and profile of the requiring service. Bandwidth, latency, reliability, spectral and energy efficiency are part of the network characteristics the network slice instance as formulated in [46]. The model proposed in this research forms part of the Figure 3.3.2

as highlighted in the brown area. The network traffic profile and characteristics of the application service requiring network slice is classified using machine learning algorithms as proposed and forms the network slice descriptor template. The architecture continually learns from new requests and predicts service requirements using a machine learning algorithm as shown later in this study. These are then passed through admission controller which sends the information to the resource manager and if admission is granted, schedules the slice on demand. In this research, the proposed model analyses the traffic profile, classifies the subslice to service chain network functions efficiently, formulates the slice blueprints and provides all information required by the admission controller to make decisions on the resource manager and schedule slices accordingly. The automation of the process from analysing the traffic profile data to formulating the slice blueprints forms part of the enhanced subslice model.

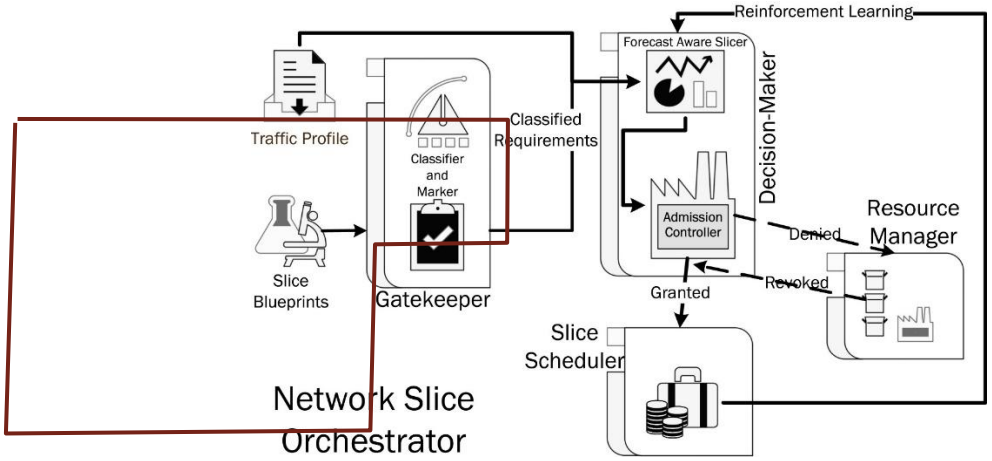


Figure 3.3.2: Network resource management and sharing architecture [45]

The whole process is self-organising and automated. The machine learning algorithm that best fits the monitored network is proposed by the ML pipeline model. For the classification of requirements shown in the Figure 3.4.2 above, the selection of the best fit model for classification is achieved with the use of LazyPredict which analyses over twenty-nine machine learning algorithms and choses the best fit. This is important as the proposed model is dynamic and should adapt to varying types of network data. LazyPredict is a python library used for hyperparameter tuning and automated model selection. Other automated model selection libraries such as Tree based Pipeline Optimisation Tool (TPOT) which uses genetic programming for machine learning pipeline optimisation were also evaluated. AutoML libraries such as H20.ai and AutoGluon were also evaluated due to their less resource intensive requirements. For the proposed model, LazyPredict is used as the default; however, It is configurable model and can be reconfigured to use AutoML if preferred due to its less computational cost compared to the more exhaustive model such as LazyPredict. Random Forest was the proposed machine learning algorithm by LazyPredict and is used as default for classification. This is automated and will differ based on specific network requirements. The machine learning pipeline model will auto select the best fit model. The Random Forest (RF) is used for clustering into subslices and types of applications however the framework can be used with any network data and suitable ML algorithm used. Reinforcement learning (RL) is used for NS aware and forecasting to reduce the risk of NS rejection. An optimised subsliced NS is orchestrated with modelled clusters sharing available physical resources and NS. The model is then trained, and trend monitored using SHAP value or Local table Interpretable Model-agnostic Explanations (LIME) machine learning algorithm. With the

successfully trained model and best classification model auto selected by the pipeline, the data is then classified to understand the application needs using the proposed machine learning algorithm and the slice efficiently allocated to each cluster of applications. Available network resource is also checked against demand and efficiently allocates the network resource to ensure good user experience and reduce the risk of denial of request. An alternative best fit network slice is orchestrated dynamically and efficiently in the event there is not enough resource to meet the need of the best fit evaluated. This forms the slice templates used for orchestration. Subscription to the Network Data Analytics Function (NWDAF) also provides relevant information about the available network resources which serves as an input to the formation of slice template in selecting the alternative best fit network slice as required. It sends alerts with intelligent optimisation information to administrators and network operators which can be used to retune the architecture as desired. The data required to provide visualizations for trend analysis was also part of the output.

Each component of this study is further discussed in Chapters 4,5 and 6. Figure 3.3.3 shows the relationship between the key components in this study. Real time data is being collected over the enterprise network. Historical and augmented data are also piped through the eSS model for analysis, resource allocation and proactive prediction of the resource needs. Mobile network providers then leverage on the inbuilt message bus in the ONSSO framework to gain real-time access to network resource requirements in real time. These streamlined processes help improve QoS and QoE.

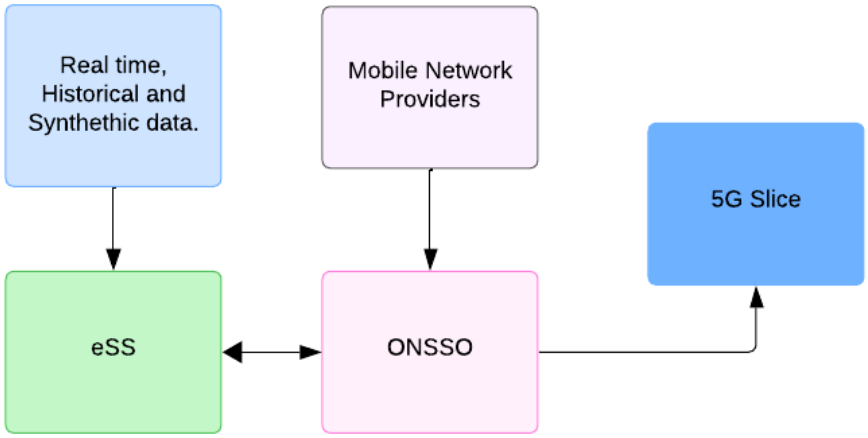


Figure 3.3.3: Flowchart showing relationship between subjects in this study.

4.0 Proposed Vertical Integration – Company Network as a Service (CNaaS)

With the rise in applications deployed in an unknown and uncertain environment with dynamic requirements across multiple infrastructures or domains with 5G network slices needs, efficiently allocating the required network resource remains a challenge. Orchestrating any of the suggested uRLLC, eMBB, or mMTC by 3GPP to meet the need of the applications on a monitored network could result in overprovisioning thereby leading to denial of a network slice for some applications. One of the ultimate goals of this study is to propose a novel vertical integration CNaaS that reduces the operational cost and capital expenditure by efficiently orchestrating network slices based on network programmability and cloud computing concept. The smart CNaaS is characterised by streamlined workflows and automated processes of network providers which further leads to lower operating expenses, as organisations can do more with fewer resources. Orchestrating a 5G network slice efficiently to an user device has been a challenge. This novel vertical will not only implement the proposed eSS model to efficiently allocate 5G network slices to an user device but also enable the automation of the optimised allocation of 5G network resource across an entire enterprise network. The modularisation of network functions in 5G network slicing serves as a key enabler to enhancing the capability of composing and defining a specific network characteristic based on service requirement. The proposed vertical integration, CNaaS is discussed in this research as a Smart Company. This is an end-to-end automated network subslice orchestration across a company's dynamic network. The end-to-end automaton capability ensures that security policies are applied consistently across all slices and simplifies the enforcement of compliance policies. CNaaS leverages on the proposed Optimised 5G Network Sub Slicing Orchestration (ONSSO) framework. This adaptive framework allows organisations to scale their network slices on-demand without significant manual intervention by integrating the proposed Enhanced Subslice (eSS) into the framework. The eSS is an intelligent network subslice model which uses artificial intelligence to understand the network, the live applications on the network with their needs and efficiently orchestrate a slice to meet their requests. The ONSSO consumes the eSS and proposes a holistic framework to manage and orchestrate the network slice across a company network. Chapter 5 and 6 will further discuss the design and architectures of ONSSO and eSS respectively. Figure 4.1 shows an overview of the eSS ML-based model's integration into the ONSSO framework.

The general objective functions considered in this study to optimise the subslice network orchestrations and which are key characteristics of CNaaS are [34]:

- The maximisation of network slices throughput.
- The optimisation of network resource ration
- The optimised allocation of network resources across each layer and the physical resource block ensuring that there is enough resource for all network slices as prioritised.
- The intelligent orchestration and allocation of alternative best slice to improve quality of service and user experience across the entire network.

The structural architecture of the proposed virtualised smart company network service presents a completely decoupled, flexible and dynamic structure which is the foundation of a fully smart software defined network. The ability to onboard or offboard a department on demand is made possible with this smart company vertical due to its capability of live monitoring and orchestrating network slice on an unknown and dynamic defined network connection service. In a service where only used network

slice resource is paid for, the cost savings will be significant as companies will only pay for network slice required and used. Only active applications on the network requiring 5G network slice will be efficiently allocated 5G network slice. The pipeline dynamically orchestrates network slice on-demand. The available network slice for that company will be efficiently orchestrated and allocated to only the departments who require network slice based on active applications on the network. The current challenge of overprovisioning which leads to wastage of resources coupled with the challenge of underprovisioning which leads to performance degradation is reduced with the capability of efficiently orchestrating network subslices to utilise only the user equipment required resource. The goal is to maintain performance and good quality of service (QoS) across diverse use cases within the organisation, especially as traffic patterns change dynamically.

This new vertical, CNaaS, employs the proposed lean process by the ONSSO framework to transform collaboration between network providers by implementing a message bus which network providers subscribes to for real-time information. This enables the network providers to proactively identify network resource needs thereby efficiently orchestrating network slice and allocating resources efficiently. Currently, the Virtual Network Operator (VNO) must wait for the Application Provider (AP) to send the user equipment slice requirement and then form the slice template; the VNO will then send the requested slice information to the Infrastructure Provider (IP) who is waiting on VNO for this information in order to allocate the required physical resource. The IP then sends the required orchestrated 5G networks physical resources to VNO for the slice orchestration who then maps this to the demanding user equipment. The proposed Enhanced Subslice module (eSS) pushes the real time user equipment demands and the predicted slice demands onto the message bus.

Figure 4.1 shows the input, process with the respective ML algorithm pipeline proposed and predicted output of the proposed CNaaS framework. The CNaaS ML model pipeline, which is the eSS, takes the network traffic data of all the user equipment on the enterprise network as input. Real time and historical data are processed to dynamically allocate the optimised network function to be chained to meet the demand of the user requirement. The data analysis mechanism in the CNaaS also processes the policies, business goals, SLAs, and other network KPIs to formulate the network slice template. Caching is implemented to improve the performance and reduce latency in the subslice orchestration process. Logging, monitoring, and fault detection mechanism are also implemented to improve QoS and meet desired SLA. These mechanisms within the ONSSO framework are further discussed in Chapter 5. The eSS model can be reconfigured to select the most suitable machine learning algorithm based on the type of network being monitored and the dynamic environment. Available network resource is evaluated and the most suitable alternative network subslice to meet the required slice is ultimately suggested and orchestrated. The eSS ML-based model is further discussed in Chapter 6. Bespoke subslices of uRLLC, mMTC, eMBB and other bespoke network subslices are the output to meet the requirements of all user equipment requiring 5G network slice on the enterprise network.

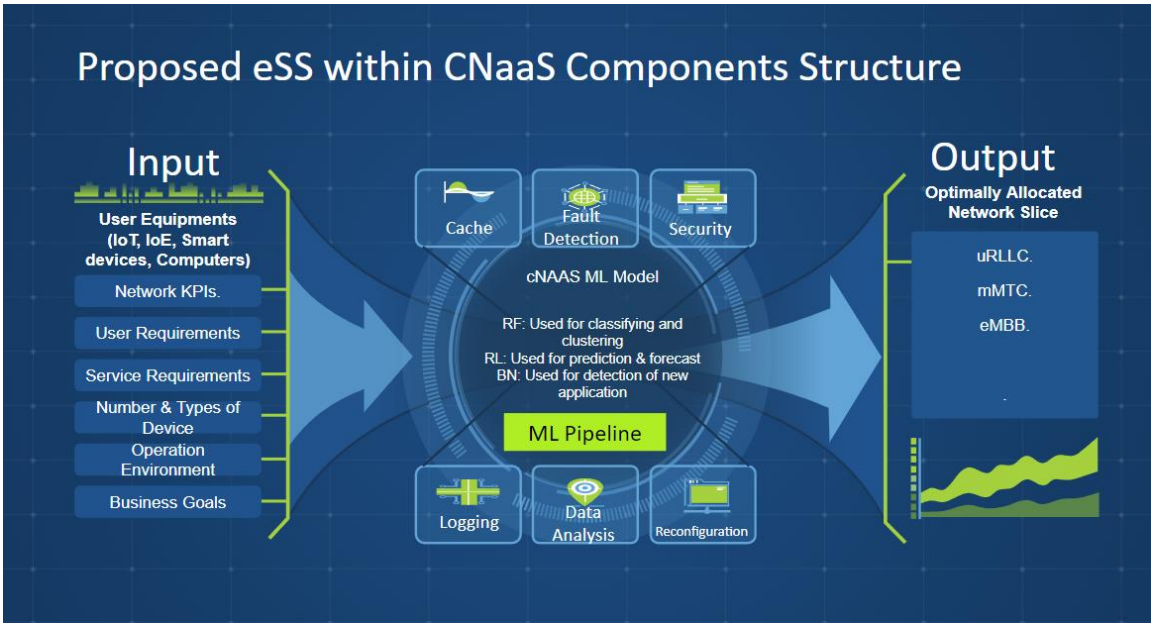


Figure 4.1: Proposed CNaaS Components Structure

A smart company network can have diverse departments. The proposed framework will scale up and down automatically as required. Similarly, this smart organisation or digital twin company will require different slices for the various departments depending on the type of tasks or services and applications that is run by the department. Figure 4.2 illustrates a snapshot of the dynamic departments structure for the proposed smart company, or the company digital twin used as a case study in this study. The organisation structure is fluid and scalable. It is flexible and the nature of service could change easily. More departments can be easily formed or disbanded and the required network slice efficiently orchestrated for the user devices available on the network.

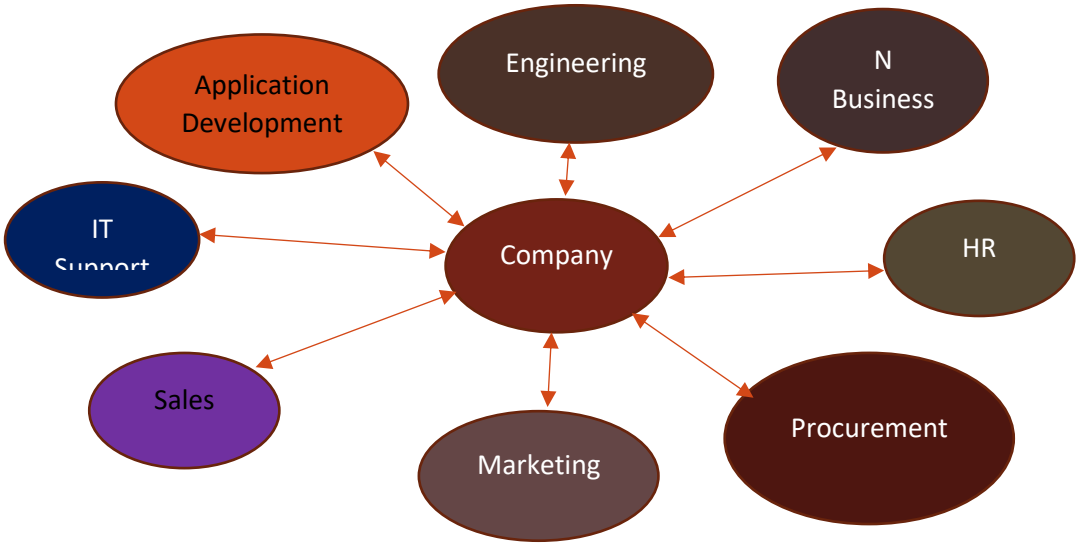


Figure 4.2: Company and its Departments overview

In a healthcare vertical, there are different types of departments in a smart hospital, such as orthopaedic, ante-natal, neurology amongst others with varying types of equipment and network service requirements. The smart hospital will require more than just a single type of network slice to fulfil the needs of all its various applications such as bench top devices for example a sphygmomanometer, pulse oximeter or a blood glucose meter which may require low latency and high reliability network slice being low compute-intensive devices while some other applications in the smart hospital which are high compute-intensive applications such as the database systems may require a network slice with high bandwidth instead. Some specialist treatment clinics might be physically located in a separate location, for example, the clinic for orthopaedic special treatment might be located in a separate building from the maternity wing. A programmable and scalable network slice would be more efficient and yield better user experience across such a diverse cross domain network.

Similarly, in a different vertical from healthcare, as in most Industry 4.0 organisation department's structure, the smart company used as a case study in this research has multiple departments providing varying services with diverse network service requirements. Departments can initiate a network slice request by sending the parameters of what they require in the form of a network template. Department such as the Marketing department who may require real time video streaming with HD quality for closing sales and making a good impression and another department which may require a reliable service with low latency and a guaranteed service will send a network request for high mobile bandwidth and ultra-reliable low latency network slice which can support the number of users connected as well as the resource being shared.

One of the departments in the proposed smart company used as a case study in this research is an application development team. The team could work closely with another team in the smart company such as the research and development team in the proposed smart company to develop massive, connected sensors. These sensors could then communicate by machine to machine, and some are machine to human communication. A department which is also part of the smart company regularly sells over the air, demoing the products and advertising over the internet. The smart company also has a department which manages sensitive and time critical solvents and their logistics. It continually monitors the trucks carrying these highly sensitive solvents within the warehouse in real time. Figure 4.3 illustrates an overview of the multi-service smart company structure. These teams could co-locate or be physically located in multiple buildings or locations. The architecture is flexible and decoupled.

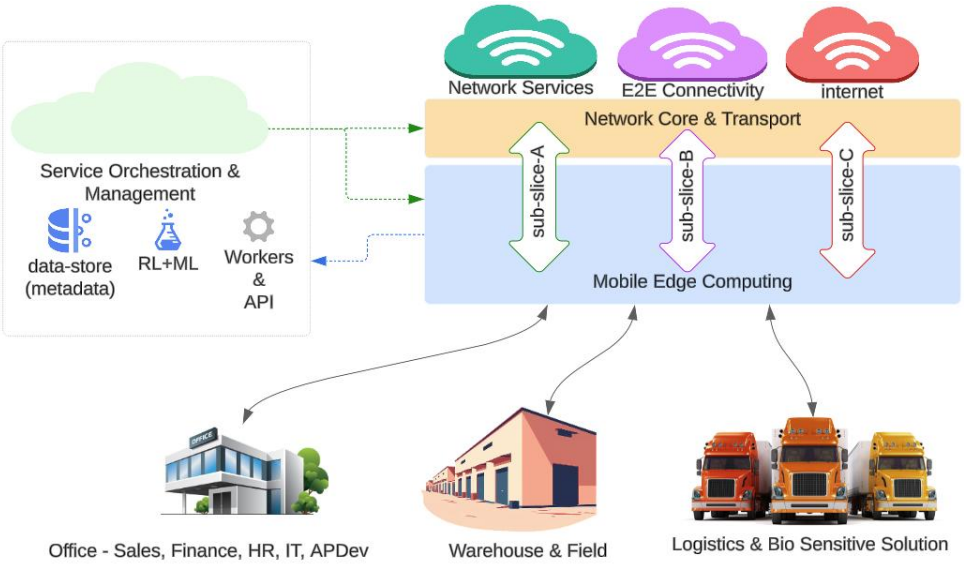


Figure 4.3. Smart Company Structure

The Smart company organisation structure in terms of location is shown in Figure 4.3. The headquarters or head office host various departments such as the marketing department, Information Technology (IT) support department, Human Resources (HR) department and a host of other departments who have diverse and varying network needs. Most of the services run at the headquarters require to be available, reliable with high throughput. The smart company also has another location with some other departments such as the application development department, the engineering department, and the operations department. These are located in the warehouse space. The warehouse has loads of connected sensors and its requirements for network varies compared to the requirements at the headquarters. Two different locations also require a low latency network to monitor and track mobile trucks within and outside the warehouse carrying bio sensitive solutions. All these locations share the same service provider and same charging subscription. The service provider charges for a single slice per smart company. The single slice is efficiently shared and orchestrated by sub-slicing the available network slice. This is done dynamically thereby reducing latency and saving the users cost.

The resource available on each network slice is limited and it is imperative that the resources are managed efficiently for optimised performance [16]. Table 4.1 shows the various key network slice requirements of the departments in the proposed smart company model. It demonstrates that various applications within the company require network slices with varying characteristics. Figure 4.4 illustrates the lifecycle overview of the slice. The end-to-end network slice orchestration and sub slicing in the CNaaS vertical goes through four major phases which are:

- The design and planning phase,
- The construction and initiation stage,
- The management and monitoring phase
- As well as the decommission and revoke phase.

The process and templates used in each phase are further discussed under the ONSSO in Chapter 5. The KPIs, SLAs and other required network features are gathered and efficiently grouped into modular structures to form sub-subsliced network, subsliced network and network slice in the preparation

stage. The eSS model is designed to facilitate and automate the design and planning phase of network subslice optimised orchestration. Within this machine learning based eSS model, the monitored network data is analysed and evaluated to form parts of the requirement gathering to build the network slice template. Performance metrics are evaluated and defined company policies such as service level agreements, security policies, admission, and access control policies all form part of the eSS model input. The eSS model uses machine learning to understand the application needs and efficiently allocates resources as required or allocates alternative best fits to manage limited available resources thereby improving users experience and reducing the risk of service denial. The eSS helps generate information that creates the slice templates specifying the required network functions, compute, storage, virtualised infrastructure components, network capacity and service chaining logic for the required network sub-slice. Validation, simulation, and continuous learning are designed as part of the eSS model.

Following the design and planning phase with the use of proposed eSS model, the focus shifts to the initiation and deployment of the network subslices. VNO, MNO, ASP and Infrastructure Providers subscribe to the eSS and obtain information from the generated slice template to provision and deploy the defined underlying infrastructure required such as servers, virtual machines, storage, compute, network capacity, switches, containers and other required components to orchestrate the characteristics of the network subslice. Following the provisioning of the physical and virtual infrastructure, the service and application within each network subslice such as network function instances, connectivity, security and policies applications are then configured. The slice then gets instantiated, the service slice template is used for configuration and the slice gets activated in the instantiation stage of the slice lifecycle. Once the slice is activated, it gets deployed to the associated application using the rules defined in the Network Slice Subnet Selection Policy Function (NSSP).

In the third stage of its life cycle, the fully operational network slice at this stage is continually monitored and optimised. With the proposed ONSSO framework and the eSS model, collaboration amongst all stakeholders such as the end users or equipment on the enterprise network, the VNO, the MNO, ASP and Infrastructure Providers is smoother thereby facilitating continuous monitoring and management by all as they subscribe to the eSS model. Service assurance activities are effectively executed by all stakeholders as they detect and resolve faults, monitor KPIs and enforce policies such as security policies.

The slice then finally moves to the last stage where the network resources are decommissioned and deactivated. Network resources are released back to the resource pool and available for another device on the enterprise network that might require it. Data can then be securely cleaned up.

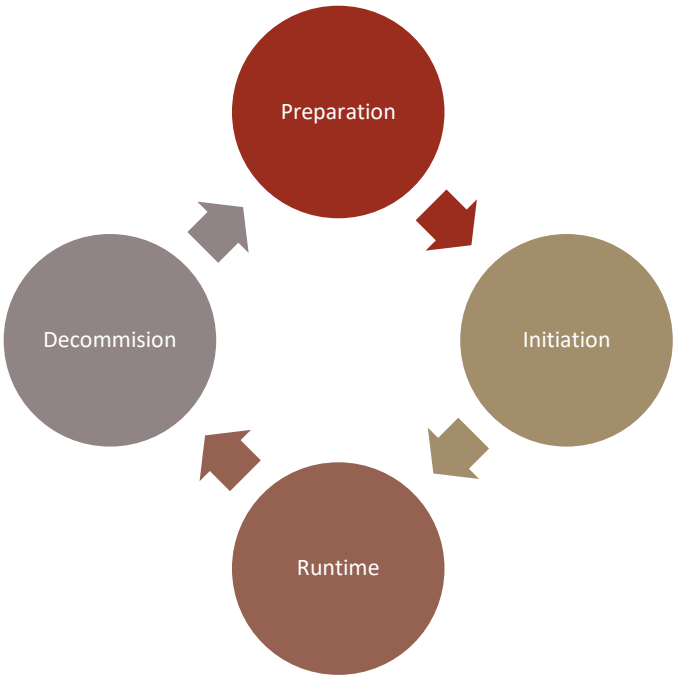


Figure 4.4: Slice lifecycle

Table 4.1. Sample Departments with Network Slice Requirements

Company		
Department	**Function**	**Network Requirement**
Engineering	Manufacturing/Engineering	• High throughput • Reliability • Low latency
N Business Units	Various business units within the company	• Medium throughput • Reliability • Low latency • Low jitter
Human Resources (HR)	Serving all business units and managing resources within the business unit	• Reliability • Low latency
Procurement	Manages all the purchases within the business.	• Reliability • Low latency • High throughput
Marketing	High-Definition video for live streaming, marketing various business products, conferencing	• Very high bandwidth • Reliability • Low latency • Low jitter
Sales	Make and receive lots of sales calls	• Coverage • Mobility

IT Support	Providing technical support to all the departments and managing the networks	• Reliability • Low latency
Application Development	Build, develop, deploy, and manage massive, connected device applications.	• Ultra-dense connectivity • Ultra-low latency • Low bandwidth • Low throughput

Departments do not have to be restricted to being physically located within the same location to benefit from the proposed Company Network as A Service (CNaaS). Communication can be made between physically separated departments using massive MIMO links and the private network slice efficiently distributed and allocated. For network slicing that utilises a pay for service user policy, this architecture will drastically save cost as the network slice will only be allocated based on the network slice required, requested, and used.

An overview of the process flow architecture from the departments network resource requests to the network slice is represented in Figure 4.5. This also shows how the edge layer, the machine learning algorithm and management application service are deployed. This hosts the network slice management service. The cache service proposed is also deployed in this edge layer with the slice management service. This will further improve latency and performance.

The physically separated and collocated departments sharing a private network slice will initiate the network request with descriptions of the type of service required in the user subscription profile. This is the network slice template and can be sent via an Application Programming Interface (API) endpoint over a TCP, IP or UDP protocol. The network slice template will contain all the packet information, slice time, allowed latency, Service Level Agreement (SLA) policy violation penalty, expected throughput, central processing unit (CPU utilisation), memory reserved, disk utilisation, Graphics Processing Unit (GPU) engine utilisation and the quality of service (QoS) requirements. The request goes to the edge layer and Chapter 6 discusses the edge layer in more detail. The controller also issues resource assignments and monitors the activities of the network resource utilisation. The RAN, transport and core are configured and allocated as required end to end as the S-NSSAI maps each slice to the corresponding user equipment.

The proposed vertical represents a significant advancement in adaptive end-to-end 5G network slice orchestration management. By leveraging the ML-based pipeline model for orchestration of 5G network slices, the CNaaS can achieve levels of efficiency, scalability, and reliability previously unattainable. As this vertical continues to evolve, artificial intelligence (AI), 5G technologies and beyond will play a central role in shaping its future, providing new opportunities for innovation, growth, and improved user experience.

The remainder of this study will discuss the process flow in the proposed smart company network, how the service chaining function helps to integrate the various network functions, the KPIs considered, the communication network and proposed eSS and ONSSO network architecture. The results are then evaluated and areas of future study recommended.

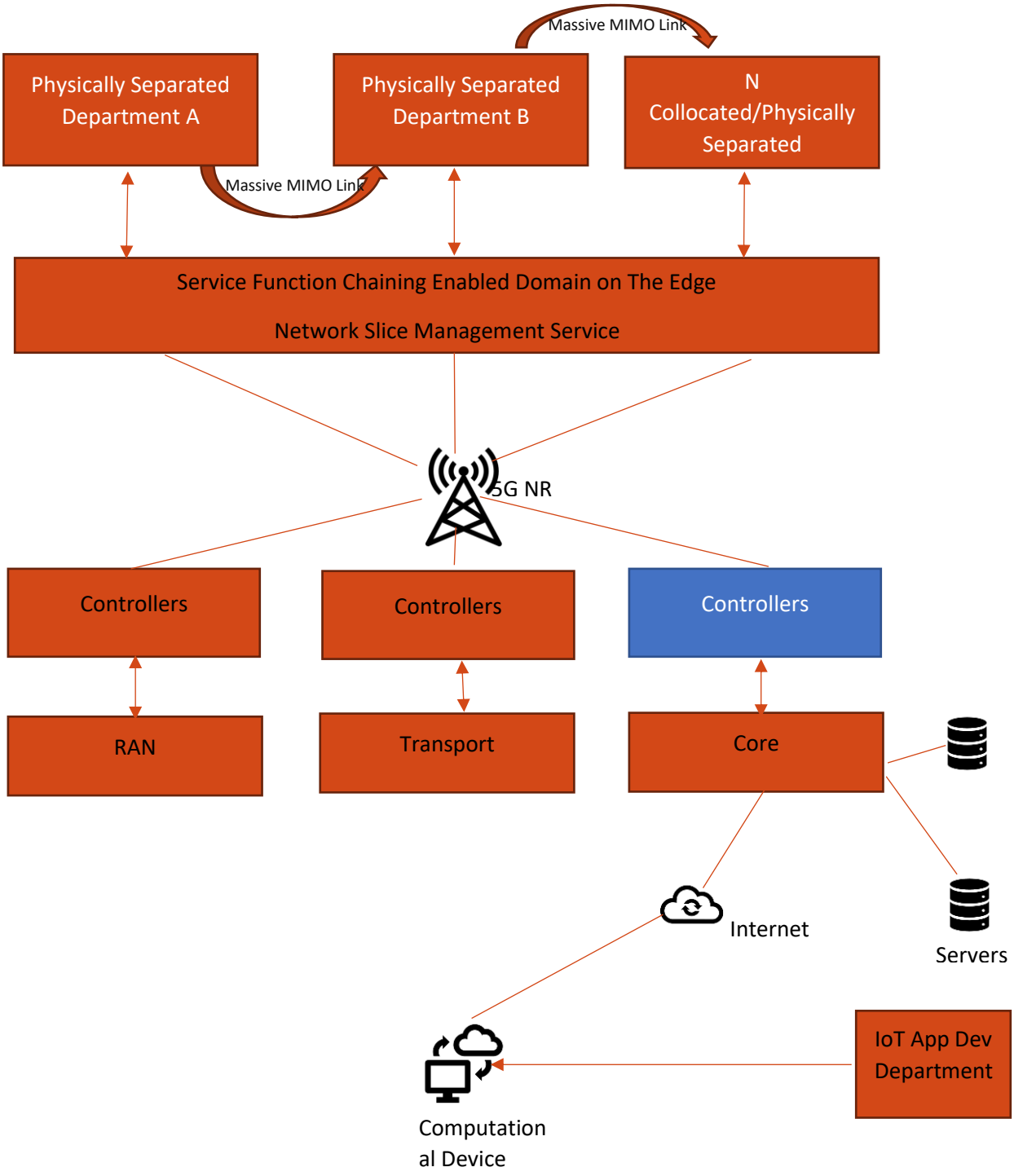


Figure 4.5: Process Flow Architecture

5.0 Proposed Framework – ONSSO

5G promises to be flexible however, due to limited resources, the full advantage of 5G has not been seen. There is a need for a flexible and adaptive framework to enable the implementation of the 5G promises. A framework that optimises the orchestration of 5G network slice right from when an application is detected and monitored right through the orchestration and deployment of the 5G network slice. The current process is dependent on multiple types of mobile network providers. There is need to streamline and automate this process, The proposed ONSSO framework is modelled to holistically manage 5G network sub slicing management, optimised orchestration, security, fault tolerance, comprehensive 5G network data analysis, self-organising, increased availability and ability to meet quality of service requirement. This is achieved with the implementation of service function chaining and continuous improvement of the network slice orchestration. The framework achieves an efficiently orchestrated subslice on a company network by continuously monitoring the network resources and continuously learning from the network data using machine learning. It learns from a combination of the real time and historical network traffic data. The subsliced network sliced into platform, application and infrastructure are further subsliced with the consideration of the following factor and metrics:

- Quality of service requirements which include subslice based on latency and reliability,
- Security requirements which include slices based on firewall policies and security level,
- Slice based on bandwidth and throughput which included slices based on high throughput for data intensive tasks and bandwidth allocation to subslices as needed. The resource type (storage and compute resources), network slicing type (eMBB, mMTC and uRLLC) and service types (Multimedia streaming, web applications, IoT Services) are also in the subslice categories derived from this research as features for further classification to efficiently orchestrate a network subslice.

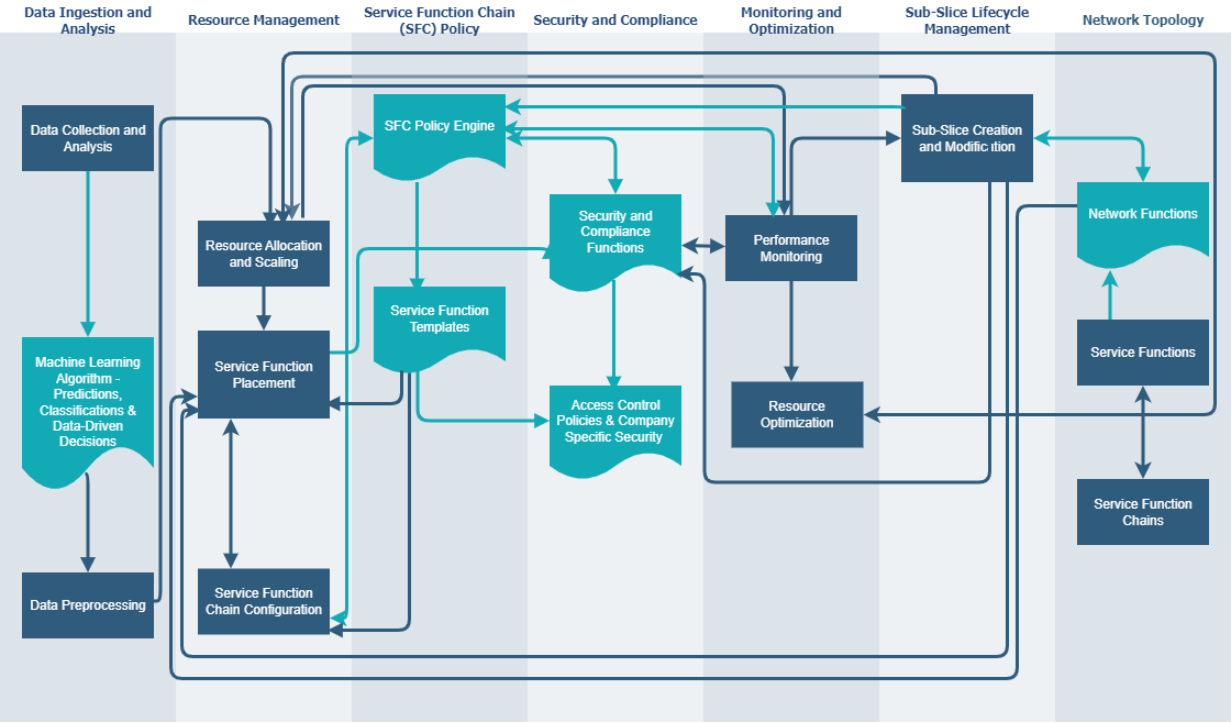


Figure 5.0. ONSSO Swimlane

The proposed ONSSO framework autonomously manages the orchestration of network subslices, ensures that security and performance requirements of subslices are met, enforces compliance, self learns from network data to continuously improve and efficiently orchestrate network subslice on demand, adapts to volatile network change demands, efficiently utilise resources in real time by using service functions and machine learning tuning. This Chapter discusses how this proposed ONSSO framework achieves these along with the pseudocode for the various components and stages in the ONSSO framework.

Figure 5.0. shows the proposed holistic representation of the framework from real time data collection, ingestion of historical datasets, through policy enforcement which forms part of the request sent to NSSF, monitoring and logging for reporting to resource allocation, modification, and service function chaining. It shows the ability for the framework to self-organise dynamically and the use of machine learning to continually gain insights and make decisions while ensuring optimised utilisation of available resources, performance to meet the quality of service and high availably to improve user experience.

Available connected devices on the company network are scanned live across all network layers in the Data Ingestion and Analysis stage of the ONSSO framework. The network data is then passed in real time through machine learning model (eSS) to obtain insights from the data to make decisions regarding efficiently orchestrating network subslice. Historical datasets are also passed through the machine learning to continually learn from past data and tune itself. The machine learning algorithm analyses the scanned data, classifies the data, uses the network data to make predictions and other data-driven decisions such as policy decisions, subslice requirements and resource allocations to efficiently orchestrate a subslice within the company network. The framework preprocesses the data and engineers the feature for sub-slicing. Based on the result from this research data, the default features used in the ONSSO framework for sub-slicing classification are Quality of Service Requirements (Reliability and latency), Bandwidth and Throughput (High throughput and bandwidth

allocation), Security Requirements (Firewall policies and security level), Resource Type (Storage and compute resources), Network Slicing Type (Massive Machine Type Communication, Enhanced Mobile Broadband and Ultra-Reliable Low Latency Communication), and Service Type (Multimedia Streaming, IoT Services and Web Applications). Based on the subsliced engineered features, the connected devices are classified into subsliced requirements and subslice templates are created. The slice templates include requirements such as security needs, bandwidth requirements, latency, and sensitivity.

With the subslice requirements collected and devices classified, the available resources are continuously monitored and managed to ensure efficient utilisation of the resources to meet the subslice resource requirements. In the Resource Management stage, the framework self-organises by dynamically allocating resources based on demand, access control policies and service function templates. Availability of network resources such as compute, network functions, storage, and spectrum are checked against the suitable service function template matched by the framework to the requested subslice. Decisions are made by the framework based on the policies to either scale if the resource is not available or choose another service function template that will still meet the subslice requirement with lesser resource usage. Monitored resource availability and performance reports are continually fed back into this Resource Management from the Monitoring and Optimisation stage. Resource allocation and scaling component also efficiently allocates resources by efficiently placing service functions within the network based on the input from the Service Function Chain Configuration which determines the order in which service functions are chained. The Service Function Placement component places the service function where latency will be at a minimum and efficiency at a maximum based on the information relating to subslice requirements, scaling instructions and resource allocation decisions made and sent by the Resource Allocation and Scaling component. It ensures that Service Function Placement efficiently places the service functions by taking the resource allocation to meet the need of the subslice into consideration. The Resource Allocation and Scaling component in the Resource Management phase also adjusts the resource allocation and scales dynamically as required by the Subslice Creation and Modification component.

The Service Function Chain Configuration receives policy related instructions from the service function chain policy engine to efficiently configure the sequence and order in which the service function is chained in the Service Function Chain (SFC) Policy and other necessary requirements for configuring service function chains. The insights from the machine learning model shapes the Service Function Templates which is used for the service function configuration and ensures the Access Control Policies and other necessary security policies are taken into consideration and monitored. The availability and current state of network function sent by the Network Functions component to the Service Function Placement can also influence the decision on determining the placement of service functions.

The Security and Compliance phase ensures the orchestrated subslice is compliant and enforces required security. Restrictions in access, compliance requirements and security measures are enforced in the subslice by collaborating with the Access Control Policies. The Security and Compliance Functions are continually monitored to not only ensure a secured and compliant subslice but to also monitor the performance of the subslice that the security functions as not had adverse effect on the subslice performance. The collaboration between Service Function Placement and Security and Compliance Functions ensures that the security and compliance requirements are also taken into consideration in determining the placement of service functions within the framework.

The Monitoring and Optimization phase continually monitors the performance of the subslice, and the Resource Optimization component optimises the network resource based on the performance data. The creation and modification of subslice on the network is also continuously monitored to get an insight as well as to efficiently orchestrate the required subslice.

The Subslice Lifecycle Management stage manages the creation and modification of the subslice. Right from the creation to the deletion of the subslice and freeing up of network resources for reuse. The Subslice Creation and Modification interacts with the Network Topology to determine which network function and service functions to allocate.

The Service Functions in the Network Topology are chained and follow the sequence and order of chaining defined in the Service Function Chain. They provide specific network services based on the subslice requirements. Some of these service functions are load balancer, firewall and deep packet inspection. These service functions use the underlying Network Functions to send and receive network traffic data and relies on it for network connectivity and interaction. Available Network Function are allocated to ensure optimised orchestration of the network subslice.

The following sub sections outlines the various stages with their respective pseudocodes that make up the ONSSO framework end to end as shown in Figure 5.0.

5.1 Device Monitoring, Data Discovery, Collection and Preprocessing:

Data is collected real time from all connected devices on the network at all network layers including core network, the radio access network, transport layer and management functions. This research focuses on the transport layer; however, the same framework can be used across all the network layers simultaneously. The network is scanned for active devices requesting a network slice, performance metrics, traffic pattern and device requirement is collected in real time from the dataset. The device network requirement such as latency, bandwidth, security are also collected to form the subslice template. Available and connected devices on the company network are continually monitored, and any changes in requirements are automatically detected with the user request updated in real time to support adjustment of network slices dynamically as required. The following pseudocode then generates an output of collection of network slice requests, slice template and duration as required by each device on the company network.

```
def monitor_devices():
    connected_online_devices  = discover_connected_online_devices ()
    for device in connected_online_devices :
        online_device_requirements = analyze_online_device_requirements(device)
        nsrs_new = create_network_slice_request(device_requirements)
        # Add or update NSRs based on device needs.
        update_network_slice_requests(nsrs_new)
```

Figure 5.1.1 Device Monitoring, Data Discovery, Collection and Preprocessing

5.2 Network Data Collection and Preprocessing:

The performance data collected from the network is pre-processed to remove noise and analysed to identify the specific subslice requirements on the company network and the potential service function needs. This ready pre-processed and ready for analysis network data are then fed into the default machine learning algorithm defined or tuned as researched and used in the proposed enhanced

subslice model. The pseudo code below accepts the network performance data as input and outputs the pre-processed data for learning and prediction.

```
function collect_and_preprocess_data():
    # Collect data from all network layers
    ran_network_dataran_network_data = collect_ran_network_data()
    transport_network_datatransport_network_data = collect_transport_network_data()
    core_network_data core_network_data = collect_core_data()
    management_data = collect_management_data()

    # Preprocess data from each layer
    ran_network_dataran_network_data = preprocess_ran_network_data(ran_network_data)
    transport_network_datatransport_network_data = preprocess_transport_network_data(transport_network_data)
    core_network_data = preprocess_core_data(core_network_data)
    management_network_data = preprocess_management_data(management_network_data)

    return ran_ network _data, transport_ network _data, core_ network _data, management_ network _data
```

Figure 5.2 Network Data Collection and Preprocessing

5.3 Network Data Preprocessing and Feature Engineering:

The collected network data in Realtime at all network layer is engineered to obtain network subslice features such as signal strength in radio access network, latency requirement in the transport layer, alarms in management functions and packet routing in core networks. This is fed as input at this stage of the framework as shown in the Pseudocode below. Processed network data to be used for feature engineering and modelling are then generated as output.

```
def collect_and_preprocess_data(dataset):
    raw_network_data = extract_data_from_dataset(dataset)
    preprocessed_network_data = preprocess_network_data(raw_network_data)
    return preprocessed_data
```

Figure 5.3 Network Data Preprocessing and Feature Engineering

5.4 Subslice Feature Engineering

The processed data from the company network performance dataset are then used at this stage of the framework to categorise traffic and devices into the network subslice requirements by determining the features from the historical dataset and real time data. The following Pseudocode is used at this stage of the framework.

```
def engineer_features(preprocessed_data):
    engineered_subsliced_features = create_subsliced_features(preprocessed_data)
    return engineered_subsliced_features
```

Figure 5.4.1 Subslice Feature Engineering

The engineered subsliced features are then used to obtain trained machine learning models as shown in the Pseudocode below.

```
def train_machine_learning_models(engineered_subsliced_features, labels):
    model = select_and_train_model(engineered_subsliced_features, labels)
    return model
```

Figure 5.4.2 Subslice Feature Engineering

After the feature engineering has been done, the model was trained and evaluated to evaluate its accuracy. With successful result, the metrics required for optimised QoS of a 5G network slice was then evaluated before further analysis and training. Network slice requirements are extracted such as latency, bandwidth, throughput, reliability, jitter, data rate, packet loss, packet delay, data volume, speed and required network functions are evaluated and created. The network functions are chained using service chaining and self-organising network concept.

5.5 Network Service Type Detection:

The characteristics of available devices on the company network are then classified into the key device types. The following pseudocode then yields the network service type output.

```
def classify_service_type(device):
    # Analyze device characteristics to classify service type.
    if is_ urllc_device(device):
        return "URLLC"
    elif is_embb_device(device):
        return "eMBB"
    elif is_ massive_iot_device(device):
        return "Massive IoT"
    else:
        return "Service Type Unknown"
```

Figure 5.5 Network Service Type Detection

5.6 Templates:

A library of templates required based on each company specific network needs and policies are defined Figure 5.6.1 and 5.6.2 shows the default modifiable template defined in the proposed ONSSO framework.

5.6.1 Service Function Templates:

Each service function template specifies a set of service function and the order in which they are chained as shown in the pseudocode below.

```
service_function_templates = {
    "TemplateA": ["Firewall", "LoadBalancer", "Optimization"],
    "TemplateB": ["LoadBalancer", "Firewall"],
    "TemplateC": ["Optimization"]
}
```

Figure 5.6.1 Service Function Templates Pseudocode

5.6.2 Subslice Templates:

The default subslice template in the ONSSO framework is shown below. This encapsulates the common requirements for a secure, high bandwidth and low latency subslice profile in JSON format.



Figure 5.6.2 Subslice Default Template

5.7 Service Function Chaining (SFC) Policy Engine:

The service function chaining policy that determines the order in which the template is selected from the defined library of service chain templates at the previous stage is then implemented. It determines the sequence in which service functions are chained to meet each subslice requirements. The policy also specifies alternative or a fallback template which require lesser resources and still meet the quality-of-service requirements if the available resources to meet the original requirements is not enough. The available resources are continually monitored with service functions. Once a subslice has been deleted and resources freed up, the subslice is then rechained to efficiently orchestrate the subslice based on resource availability and the network subslice requirements. In the pseudocode below, the determining_sfc_policy function returns the suitable primary service function template based on the subslice requirement and available resources. If no suitable template is found, the function then selects a suitable fallback template which requires much lesser resource and able to meet both the subslice requirements and the subslice quality of service requirements to return. The selecting_template is the function called in the pseudocode to check available resources and match to the service function template requirement. If this can be met, the template is then selected as the primary template. The selecting_fallback template function is the function called if the selecting_template couldn't return a primary template. The selecting_fallback template returns a suitable template based on available resources and additional specific subslice requirements to fulfil the main requirements of the subslice. The selecting_fallback_template uses the can_allocate_resources function to match the available resource to the template requirements and determines whether the resources can be allocated. The criteria for selecting the fallback template is defined in the meet_fallback_criteria based on company's subslice requirements. This is based on specific company network policy needs.

```python
# ONSSO Framework Service Function Chaining (SFC) Policy Engine
def determining_sfc_policy(sub_slice_requirements, available_resources):
    # Logic to select an appropriate service function template
    selected_template = selecting_template(sub_slice_requirements, available_resources)

    if selected_template is not None:
        return service_function_templates[selected_template]
    else:
        # If no suitable template can be found, select a fallback template
        fallback_template = selecting_fallback_template(sub_slice_requirements)
        if fallback_template is not None:
            return service_function_templates[fallback_template]
        else:
            # If no suitable templates are available, return an empty list or handle the situation as needed.
            return []

# Function to select the most suitable template based on requirements and available resources
def selecting_template(sub_slice_requirements, available_resources):
    for template_name, template_requirements in service_function_templates.items():
        # Compare the resource requirements of the template with available resources
        if can_allocate_resources(template_requirements, available_resources):
            return template_name
    return None  # No suitable template found

# Function to select a fallback template when resources are insufficient for the primary template
def selecting_fallback_templatse(sub_slice_requirements):
    for template_name, template_requirements in service_function_templates.items():
        # Logic to select a fallback template based on sub-slice requirements
        if meet_fallback_criteria(template_requirements, sub_slice_requirements):
            return template_name
    return None  # No fallback template found

# Function to check if resources can be allocated for a given template
def can_allocate_resources(template_requirements, available_resources):
    # Compare resource requirements (e.g., CPU, memory, storage) with available resources
    # Implement the logic to evaluate if resources are sufficient for the template
    if template_requirements['CPU'] <= available_resources['CPU'] and \
       template_requirements['Memory'] <= available_resources['Memory'] and \
       template_requirements['Storage'] <= available_resources['Storage']:
        return True
    return False

# Function to define criteria for selecting a fallback template based on sub-slice requirements
def meet_fallback_criteria(template_requirements, sub_slice_requirements):
    # Implement the logic to evaluate if the fallback template meets specific criteria
    if sub_slice_requirements['LatencyRequirement'] == 'Low' and \
       template_requirements['LatencyRequirement'] == 'Medium':
        return True
    # Add more criteria as needed
    return False
```

```python
# Example of available resources
available_resources = {
    'CPU': 16,       # Available CPU capacity
    'Memory': 64,    # Available memory capacity (in GB)
    'Storage': 500   # Available storage capacity (in GB)
}

# Example of sub-slice requirements
sub_slice_requirements = {
    'LatencyRequirement': 'Low',
    'BandwidthRequirement': 'High',
    'SecurityLevel': 'Secure',
    # Add more requirements as needed
}

# Determine the SFC policy and selected service function template
sfc_policy = determining_sfc_policy(sub_slice_requirements, available_resources)

# The 'sfc_policy' variable now contains the selected service function template (or fallback template)
```

Figure 5.7.1: Service Function Chaining (SFC) Policy Engine Pseudocode

5.8 Resource Scaling and Allocation:

Available resources such as network capacity, CPU and memory are dynamically scaled using cloud technologies and allocated to accommodate the service function requirements and the changing network subslice demands. This framework stage scales and accommodates the network resources catering for the network functions and service functions to ensure the resource intensive network

tasks are catered for as well as other processing and forwarding requirements of the network traffic. The pseudocode below shows the functions defined within this framework to scale and allocate available resources to meet the subslice demands. allocate_resources function calculates the total resources required by the service functions and then calls the can_allocate_resources to determine if there is enough available resource to allocate if not, scaling is implemented, and the additional resources are allocated as required, or available resource utilisation is optimised depending on defined policy.

```
# Function to allocate and scale resources for a sub-slice within the ONSSO framework
def allocate_resources(sub_slice_requirements, service_functions, available_resources):
    # Calculate the total resource requirements based on service functions
    total_requirements = calculate_total_requirements(service_functions)

    # Check if available resources can satisfy the total requirements
    if can_allocate_resources(total_requirements, available_resources):
        # Allocate resources to each service function based on their requirements
        allocate_individual_resources(service_functions, available_resources)
    else:
        # Implement scaling logic to allocate additional resources or optimize resource utilization
        scaled_resources = scale_resources(total_requirements, available_resources)

        # Allocate scaled resources to service functions
        allocate_individual_resources(service_functions, scaled_resources)

# Function to calculate the total resource requirements based on service functions
def calculate_total_requirements(service_functions):
    total_requirements = {
        'CPU': 0,
        'Memory': 0,
        'Storage': 0,
        # Add more resource types as needed
    }

    for function in service_functions:
        for resource_type, requirement in function['ResourceRequirements'].items():
            total_requirements[resource_type] += requirement

    return total_requirements

# Function to allocate resources to individual service functions
def allocate_individual_resources(service_functions, available_resources):
    # Implement the logic to allocate resources to each service function
    for function in service_functions:
        function['AllocatedResources'] = {
            'CPU': allocate_cpu(function['ResourceRequirements']['CPU'], available_resources['CPU']),
            'Memory': allocate_memory(function['ResourceRequirements']['Memory'], available_resources['Memory']),
            'Storage': allocate_storage(function['ResourceRequirements']['Storage'], available_resources['Storage']),
            # Add allocation logic for more resource types as needed
        }
```

```
# Function to check if available resources can satisfy requirements
def can_allocate_resources(requirements, available_resources):
    # Implement the logic to check if available resources meet the requirements
    return (
        requirements['CPU'] <= available_resources['CPU'] and
        requirements['Memory'] <= available_resources['Memory'] and
        requirements['Storage'] <= available_resources['Storage']
        # Add more resource checks as needed
    )

# Function to scale resources based on requirements and available resources
def scale_resources(requirements, available_resources):
    # Implement the logic to scale resources (e.g., allocate additional resources or optimize utilization)
    scaled_resources = {
        'CPU': calculate_scaled_cpu(requirements['CPU'], available_resources['CPU']),
        'Memory': calculate_scaled_memory(requirements['Memory'], available_resources['Memory']),
        'Storage': calculate_scaled_storage(requirements['Storage'], available_resources['Storage']),
        # Add scaling logic for more resource types as needed
    }
    return scaled_resources

# Function to allocate CPU resources
def allocate_cpu(requested, available):
    # Implement the allocation logic for CPU resources
    # (e.g., allocate requested CPU or a fraction of it)
    return min(requested, available)

# Implement similar functions for memory and storage allocation
```

Figure 5.8: Resource Scaling and Allocation Pseudocode

5.9 Service Function Placement

This stage of the framework efficiently places the service functions within the network to reduce latency and increase efficiency. The proximity of the service functions to the destination and traffic source are considered to minimise delays. The determining_placement function in the pseudocode below checks the function in each library of service functions and determines the optimised location for the function. The finding_optimised_location is called to analyse the network topology and efficiently determines the optimised location for deploying network service functions based on proximity, latency, and capacity.

```
# Function to determine the optimal placement of service functions for the ONSSO framework
def determining_placement(service_functions, network_topology):
    placement_info = {}

    for function in service_functions:
        # Implement logic to find the best location for each service function
        placement_info[function['Name']] = finding_optimal_location(function, network_topology)

    return placement_info

# Function to find the optimal location for a service function
def finding_optimal_location(function, network_topology):
    # Implement the logic to analyze network topology and find the best location
    # for deploying the service function based on latency, proximity, and capacity.
    optimal_location = network_topology.finding_optimal_location(function)
    return optimal location
```

Figure 5.9 Service Function Placement Pseudocode

5.10 Service Function Chain Configuration

The framework configures the network traffic to follow the sequence of service functions chaining for each subslice as defined by the service function chaining policy. It configures the service forwarding between two service functions as shown in the pseudocode below:

```
# Function to configure the network for Service Function Chaining (SFC)
def configure_sfc(sub_slice_id, service_functions):
    for i in range(len(service_functions) - 1):
        # Configure the network to forward traffic from one service function to the next
        configure_forwarding(sub_slice_id, service_functions[i], service_functions[i + 1])

# Function to configure forwarding between two service functions
def configure_forwarding(sub_slice_id, source_function, destination_function):
    # Implement the logic to configure network devices (e.g., routers or switches)
    # to forward traffic from the source function to the destination function
    network_device.configure_forwarding(sub_slice_id, source_function, destination_function)
```

Figure 5.10 Service Function Chain Configuration Pseudocode

5.11 Dynamic Slice Orchestration:

Following the automated discovery of the available devices on the company network, the orchestration component of the ONSSO framework carries out checks to identify availability of network functions that could be assembled to meet the devices requirements, allocates the network functions to make up the network slices and installs the slices. The pseudocode below will take the collection of network slice requests with available resources as input and outputs an optimised on-demand provisioning of network slice.

```
def allocate_sub_slices(sub_slice_requests, available_resources):
    for sub_slice_request in sub_slice_requests:
        if can_accommodate(sub_slice_request, available_resources, current_load):
            allocate_resources(sub_slice_request, available_resources)
            install_sub_slice(sub_slice_request, allocated_resources)
        else:
            handle_insufficient_resources(sub_slice_request)
```

Figure 5.11 Dynamic Slice Orchestration

5.12 Network Resource Allocation using Machine Learning Algorithm - Reinforcement Learning:

Reinforcement learning is used in the framework to automatically learn the optimised orchestration of subslice network resources in order to continuously achieve optimised subslice network resource for the connected device. The pseudocode below accepts the processed data and resource availability as input then outputs the network resource allocation policies:

```
def optimize_resource_allocation(preprocessed_data, available_network_resources):
    allocation_policy_optimal = train_rl_model(preprocessed_data, available_network_resources)
    return allocation_policy_optimal
```

Figure 5.12 Network Resource Allocation using Machine Learning Algorithm - Reinforcement Learning

5.13 Subslice Optimisation Policies:

The framework applies the optimisation policies after the required service type has been detected and allocates resources to the subslice dynamically. The subslice policy continuously monitors the resources allocations and make adjustments depending on the quality-of-service requirements as well as the performance data available in order to efficiently and dynamically cater for the diverse network resource needs of the connected devices on the company network such as high speed broadband applications, massive data IoT applications and low latency critical applications. The pseudocode for this stage in the framework accepts the optimisation policies as defined by the company and the performance data from the network as input and outputs an optimised network resource allocation for each identified and detected subslice.

```
def optimize_sub_slices(sub_slice_performance_data, optimization_policies):
    for sub_slice_data in sub_slice_performance_data:
        optimization_policy = optimization_policies[sub_slice_data.service_type]
        optimize_resource_allocation(sub_slice_data, optimization_policy)
```

Figure 5.13 Subslice Optimisation Policies

5.14 Real Time Network Sub Slicing:

The pseudocode for this stage in the framework takes the resource allocation policies and the network slice requests as input then dynamically subslices the network as output.

```
def orchestrate_network_sub_slices(network_sub_slice_requests, allocation_policy):
    for nsrs_new in network_slice_requests:
        if can_accommodate(nsrs_new, available_resources, current_load):
            allocate_resources(nsrs_new, allocation_policy_optimal)
            install_network_slice(nsrs_new, allocated_resources)
        else:
            handle_insufficient_resources(nsrs_new)
```

Figure 5.14 Real Time Network Sub Slicing

5.15 Continuous Monitoring of Connected Devices and Learning:

The network data is continuously monitored, and the generated dataset is used to continuously train the reinforced learning model. The subslice performance, security and resource utilisation is continuously monitored using service functions and the framework dynamically optimises the service function chaining and network resource allocation in real time. The following pseudocode takes the network slice and the generated performance dataset as input then continually optimises the subslice orchestration as output.

```
def monitor_and_learn(network_sub_slices, performance_data):
    for subslice in network_sub_slices:
        collect_performance_data(subslice, performance_data)
        adapt_subslice_configuration(subslice, performance_data)
```

```
while True:
    monitor_sub_slice_performance()
    optimize_resources()
```

Figure 5.15 Continuous Monitoring of Connected Devices and Learning Pseudocode

5.16 Fault Detection with Behaviour and Security Analysis:

This stage of the framework ensures the monitored company network security and reliability. The K-Means machine learning model is used in this framework for behaviour analysis and automatically detects anomalies or faults on the company network. The processed data and identified engineered subslice features are fed as input at this stage. Any anomalies or fault in the network are detected as input. The service function is used to enforce compliance requirements and security policies. The following Pseudocode is used at this stage of the framework.

```
def detect_network_faults(preprocessed_data, engineered_features):
    clusters = apply_clustering_algorithm(engineered_features)
    identified_faults = analyze_clusters(clusters)
    return identified_faults

function detect_and_analyze_faults(data):
    # Detect faults in each network layer and management functions
    ran_network_faults = detect_ran_network_faults(data.ran_network_data)
    transport_network_faults = detect_transport_network_faults(data.transport_network_data)
    core_faults = detect_core_faults(data.core_data)
    management_faults = detect_management_faults(data.management_data)

    # Analyze and report detected faults
    analyze_faults(ran_network_faults, transport_network_faults, core_faults, management_faults)

    # Take appropriate actions based on fault analysis
    handle_faults(ran_network_faults, transport_network_faults, core_faults, management_faults)
```

Figure 5.15.1 Fault Detection with Behaviour and Security Analysis

Intrusion detection is also included in the framework to cater for a secure and efficiently orchestrated network resource. The framework uses snort model and accepts the security rules and network traffic data as input. Any security threats and breaches are then detected as output of this stage as shown in the pseudocode below.

```
function analyze_security_and_detect_threats(data):
    # Analyze security at each network layer and management functions
    ran_network_security = analyze_ran_network_security(data.ran_network_data)
    transport_network_security = analyze_transport_network_security(data.transport_network_data)
    core_network_security = analyze_core_network_security(data.core_data)
    management_security = analyze_management_security(data.management_data)

    # Detect security threats
    ran_network_threats = detect_ran_network_threats(data.ran_network_data)
    transport_network_threats = detect_transport_network_threats(data.transport_network_data)
    core_network_threats = detect_core_network_threats(data.core_data)
    management_network_threats = detect_management_network_threats(data.management_data)

    # Report and respond to security threats
    report_security_analysis(ran_network_security, transport_network_security, core_network_security, management_security)
    handle_security_threats(ran_network_threats, transport_network_threats, core_network_threats, management_network_threats)
```

Figure 5.15.2 Fault Detection with Behaviour and Security Analysis

The fault detection algorithm is applied to core network, radio access network, transport layer and management function. The detected faults are analysed to predict potential impact and root cause.

5.17 Logging and Dashboard Reporting Capability:

Logs are captured within the framework at every stage and insights can be generated from the logged data. The company can create a dashboard as desired with log outputs. Service functions also report on service level agreement and key performance indicators such as packet loss, latency and other metrics defined and monitored by the company.

```
def log_and_report(sub_slice_id, logs):
    # Logic to log and report data
    log_data(sub_slice_id, logs)
```

Figure 5.17 Logging and Dashboard Reporting Capability Pseudocode

5.18 End-to-End Subslice Lifecycle Management:

Once data has been scanned in real time over the network, pre-processed, requirements gathered and passed access control, the life cycle of the instantiated subslice is managed within the framework for the creation of subslices, modifying existing subslices and deleting subslices.

```
def manage_sub_slice_lifecycle(sub_slice_id, action):
    if action == "create":
        # Logic to create sub-slice and configure service functions
        create_sub_slice(sub_slice_id)
    elif action == "modify":
        # Logic to modify sub-slice and update service functions
        modify_sub_slice(sub_slice_id)
    elif action == "delete":
        # Logic to delete sub-slice and remove service functions
        delete_sub_slice(sub_slice_id)
```

Figure 5.18 End-to-End Subslice Lifecycle Management Pseudocode

6.0 Proposed Model – eSS

The core of the ONSSO is the ML-based eSS model. The network resource block which mobile network operators can provide to meet the need of applications demanding them is limited. Allocation of these resources efficiently is currently a challenge. There is a need to efficiently allocate these limited resources to efficiently orchestrate a network slice for improved QoS and QoE. eSS aims to provide the solution to this while remaining a standalone model which can be implemented in any 5G network slice management adaptive framework.

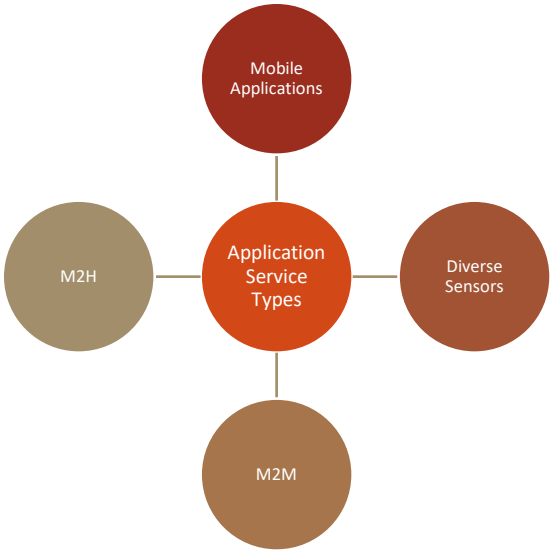


Figure 6.0.1: Types of Application Services on Smart Company Network

The Enhanced Subslice (eSS) model is a pipeline made of series of machine learning algorithms to efficiently allocate resource, chain the network resource and efficiently forecast the network slice requirement on the monitored enterprise network to reduce latency and improve both QoE and QoS. This model also includes a module to efficiently allocate the most suitable alternative slice with lesser resource requirements based on the available resources.

Network slicing alone is not enough to effectively meet the diverse service requirements of users [16]. To address this issue, the concept of network sub-slicing was explored in [16]. This approach involves grouping network slices that have similar characteristics to optimize the orchestration of network slices. In this study, sub-slicing is based on application key performance indicators (KPIs) and the network slice descriptions in user subscription profiles. This method aims to make better use of the limited available physical resources, automatically scale resources as needed, and predict potential demands for additional physical resources. These measures help prevent network slice failures or rejections and reduce the overall cost of network slicing in a dynamic, smart company network.

With the proposed Enhanced Subslice (eSS) model pipeline, network functions can be efficiently shared among isolated network subslices thereby efficiently managing, orchestrating, and allocating the available resource. The eSS allocates only the required network functions to be chained to make a 5G network slice thereby reducing the risk of over provisioning and reducing cost of allocating standard slices. The eSS model also orchestrates the best alternative fit network slice to reduce the risk of denial of slice and improve Quality of Experience (QoE). Based on the available physical resources, the eSS will analyse the best-fit slice for all live applications and predicted live applications

to come online and ensure the available resources are efficiently allocated to meet the needs of live applications. It proposes the alternative network slice with lesser network resource requirements yet still meeting the needs of the application to enhance QoS and QoE. Figure 6.0.2 describes the structure of the slices.

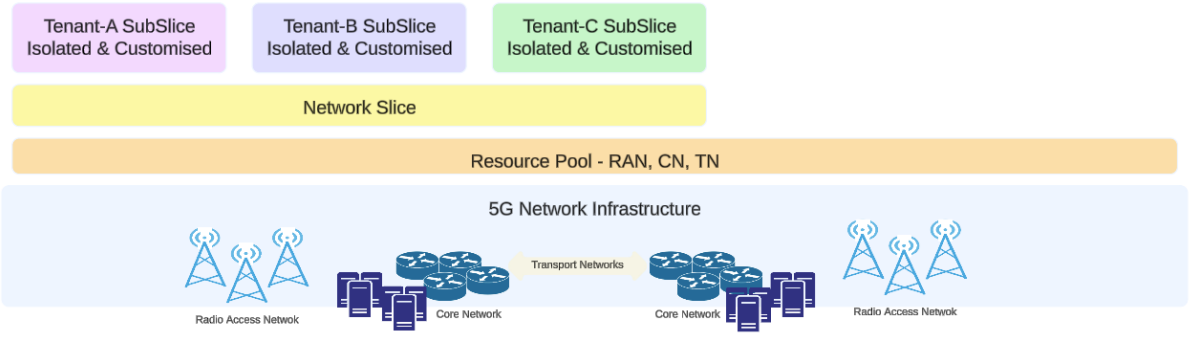


Figure 6.0.2: Network Sub Subslice Structure.

The combination of the machine learning algorithms in the eSS module pipeline aims to efficiently orchestrate 5G network slices and efficiently allocates network resources to ensure quality of service and improved quality of experience in an unknown network with changing and diverse service requirements. There are various components which facilitate the subsliced network characterisation and operation. These components are:

1) a Traffic Prediction Module which primarily leverages a reinforcement learning technique for anticipated 5G network slice traffic demand prediction and sending the traffic predictions to a slice orchestrator. This enables the proactive orchestration of the required network slice to reduce latency.

2) a Slice Orchestrator which utilises supervised learning models like Random Forest (RF) and SVM for resource allocation and then sends the resource allocation decisions to the resource allocator. It handles the classifications of the network subslice.

3) a Resource Allocator which receives the resource allocation decision from the slice orchestrator and implements it. The general orchestration and management of the orchestrated slice. An overview of these components is shown in Figure 6.0.3.

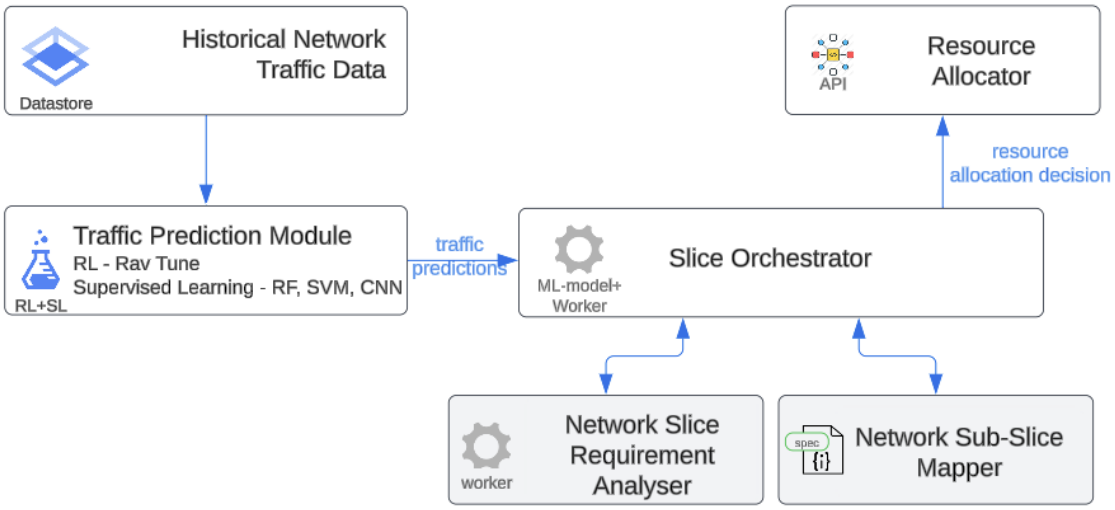


Figure 6.0.3. System architecture diagram showing the components and interactions.

The Traffic Prediction Module utilises a two-stage learning process:

- Reinforcement learning which is an offline learning that learns historical data and predicts traffic trends and anticipated demands of the 5G network slice. This component is responsible for forecasting future network traffic patterns for each network slice.

- Automated Reinforcement Learning (AutoRL), which is an online learning that continually adapts to the dynamic environment conditions and evolving user requirements. The RL agent adaptively updates its policy based on the live feedback, refines its predictions, and enables its ability to dynamically respond to the evolving traffic pattern.

A crucial aspect of the Traffic Prediction Module is the exploration-exploitation trade-offs. The module exploits its current knowledge base to make informed and accurate predictions. The module also explores new slice configurations to uncover potential improvements.

The Slice Orchestrator forms the core of the eSS module pipeline. It manages the overall 5G network slice orchestration process. The Traffic Prediction Module sends traffic predictions to the slice orchestrator and the slice orchestrator interacts with the Resource Allocator to optimise the allocation of network slice resource. The slice orchestrator consists of two subcomponents, which are:

1. The Network Slice Requirements Analyser, which forms part of the eSS with an inbuilt metrics KPI estimator and which uses LazyPredict, a python module to analyse the dataset, automatically select best fit machine learning algorithm for the given network dataset and then trains the best supervised machine learning algorithm (e.g., Random Forests, SVM, Gradient Boosting, Neural Networks). This analyses the network traffic data to identify the 5G network slice requirement for the given network pattern. It identifies the best fit supervised learning classifier (e.g., Decision Trees, Support Vector Machines) for training, feature engineering, hyperparameter tuning and selecting the model with optimised performance considering the key metrics such as latency, throughput, bandwidth, jitter, and packet loss rate.

2. Network Subslice Mapper, part of the eSS which uses bespoke inbuilt python function to select the best-fit or alternative best fit network functions configurations to be chained based on available

network resources and predicted network resource requirements for future slices. The most suitable network subslice is mapped to the applications for optimised performance considering the key metrics such as latency, throughput, bandwidth, jitter, and packet loss rate.

Figure 6.0.4 illustrates the end-to-end network slice workflow overview from application layer to the network operator layer.

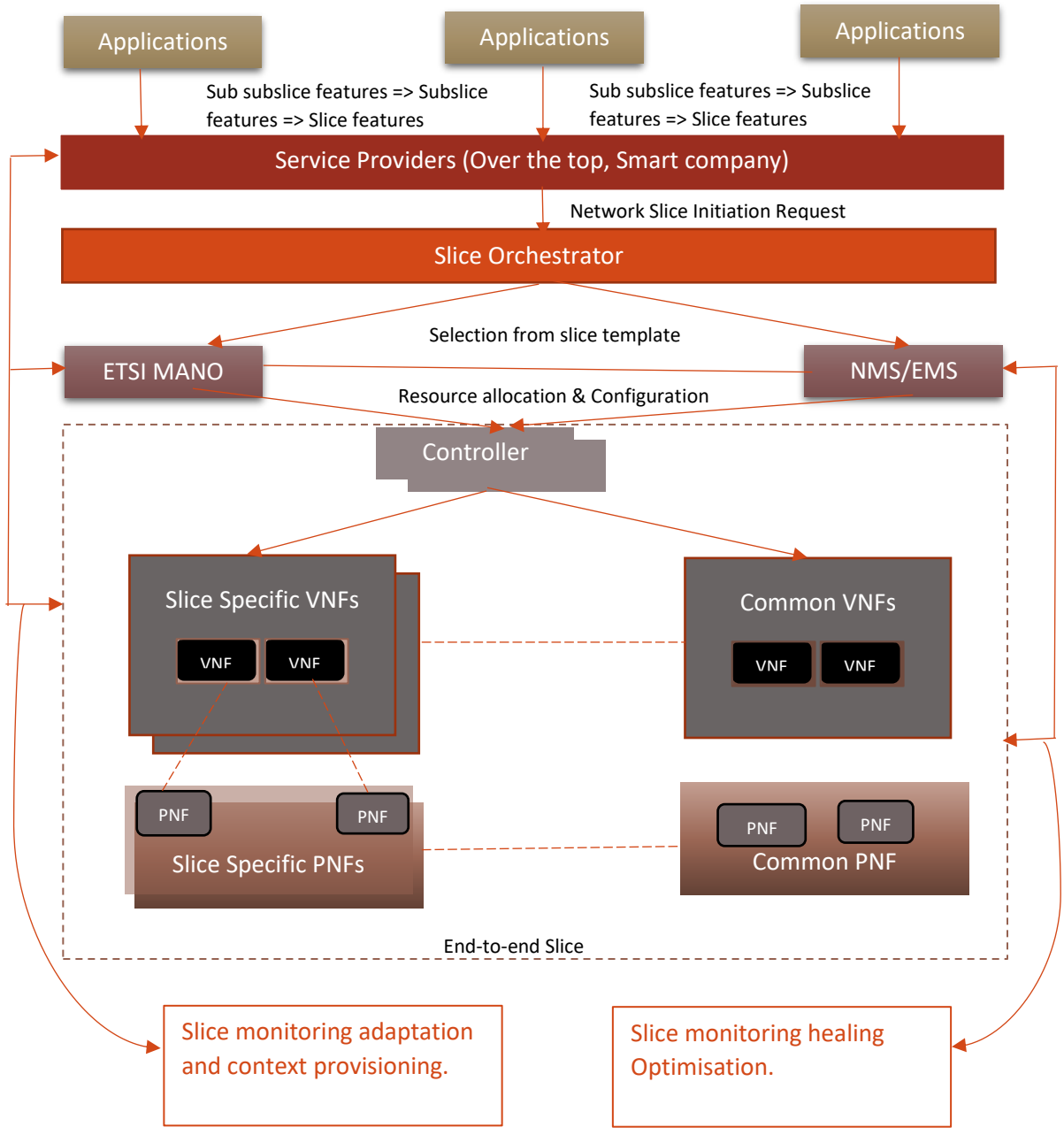


Figure 6.0.4. Smart Company End-to-end network slice overview.

Applications defined KPIs will be monitored and classified using a machine learning algorithm. The classified features will define the network characteristics which is part of the network description

template. The lower-level applications and services will be the sub subsliced of the network. Based on the type of application and service, the sub subsliced are grouped into subsliced network slice relating to platform based, application based or infrastructure-based network slice. These are then further grouped based on the 3GPP defined standard network slice and application specific required network slice resources. The service provider can then initiate the requested described network slice with required policies and deploy to various application as required dynamically. The slice orchestrator receives the request and translates that into network policies and requirements. The slice orchestrator then manages as well as requests a slice template from the European Telecommunications Standards Institute Management and Organization building blocks (ETSI MANO). Computing, storage, network function packages and resources are then allocated based on the network template. Management of the network resource lifecycle and virtual resources are handled by the building blocks. The network slice orchestrator also feeds into the Element Management System (EMS) and Network Management System (NMS) to manage the 5 main tasks identified by the ISO Telecommunications Model Framework for network management: the Fault management, Configuration management, accounting management, Performance management, and Security management (FCAPS). The performance of the network slice is monitored and fedback to improve orchestration of network slices. The network functions are configured and allocated. Some of the network functions will be slice specific while others will have general characteristics for network slices. These modular functions are chained using service function chaining. These logical network functions share the physical network function such as the router, firewall, load balancers which are purposely built for specific functions. The network functions constitute the RAN, Core and transport slice which makes up the end-to-end network slice for a service. The core network slice is made up of core network functions varying from session management, policy and charging function, mobility management, security, to forwarding of data packets. The RAN network slice is made up of network functions varying from radio resource control to packet data convergence protocol. The transport network slice is made up of network functions varying from wireless to optical segments.

The details of the type of service supported by a slice is defined in the Network Slice Selection Function (NSSF). The proposed model in this study uses machine learning algorithm to classify the features and identify the characteristics of the service supported by a slice. This is then used to form the NSSF. The Core Access and Mobility Management Function (AMF) sends a get request to the NSSF to instantiate the Network Selection Instance (NSI). Allowed AMF and Network Slice Selection Assistance Information (NSSAI) are included in the response body payload. The Single Network Slice Selection Assistance Identifier (S-NSSAI) identifies each slice and which user equipment (UE) to connect the slice to. The Slice/Service Type (SST) describes the type of service and its required information in the NSSAI. The Slice Differentiator (SD) is also specified in the NSSAI but It is an optional information. The SD identifies the different slices in the group of defined same type of slices.

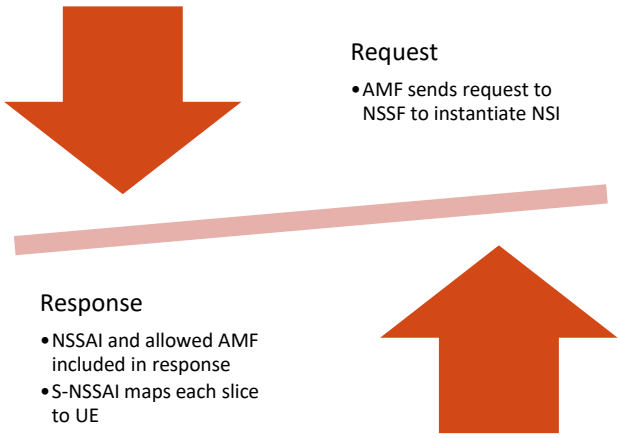


Figure 6.0.5: API call and response to instantiate network slice instance.

The main types of network functions in the 5G system architecture [69] are shown in Figure 6.0.6 below.

Network Functions	
	Radio Access Network (RAN)
	Application Function (AF)
	Core Access and Mobility Management Function (AMF)
	Data network (DN)
	Network Exposure Function (NEF)
	NF Repository Function (NRF)
	Policy Control function (PCF)
	Session Management Function (SMF)
	Structured Data Storage network function (SDSF)
	Unified Data Management (UDM)
	Unstructured Data Storage network function (UDSF)
	User Equipment (UE)
	User plane Function (UPF)

Figure 6.0.6: 5G System Architecture Main Network Function Composition

6.1 5G Network Subsliced Architecture.

The architecture of the subsliced network will be further discussed later in this study.

Table 6.1.1: Transport and core network of CNaaS

Layer	Technology/Service Type
Transport	mmWave wireless links µwave wireless links

Core Network	Infrastructure based service slice, platform-based service slice, application-based service slice are the features selected as a layer on top of the core features defined by 5G for network slices. The key features defined are uRLLC, mIoT, eMBB

This study selected features based on key KPIs to optimise the network slice by taking network slice specific objective functions into consideration while devising a general objective function for the network slices and constraints to optimise the network slice orchestration. Minimising packet delay is one of the key objective functions for ultra-reliable low latency communication to optimise the network slice orchestration. Maximising data transfer rate is another key objective function for enhanced mobile broadband slice.

The proposed eSS model uses a self-organising network concept. Six main stages were followed in this study to model the framework for the pipeline to orchestrate the network slice dynamically and efficiently for diverse applications on an enterprise network. These stages will be repeated in the cycle as new data appears to reorchestrate network slice for the new service that requires it. Figure 6.1.1 shows the process followed by stages in designing the framework. The live data is monitored, and the machine learning pipeline is used to analyse the data from the network. This is then fed to the machine learning trained model that serves as the predictor and defines the network slice template. The training of the historical dataset workflow starts with the dataset obtained from Universidad del Cauca network [42]. A summary of the steps used in formulating the pipeline are:

Step 1. Analyse the traffic profile data and Identify features or feature selection from the traffic flow dataset using -Dimensionality Reduction Algorithms (PCA, LDA). Cluster to identify number of slices the features are grouped into

Step 2. Group features with characteristics using - Clustering Algorithms (k-Means, Hierarchical Clustering)

Step 3. Group applications or sevices with similar characteristics using - Clustering Algorithms (k-Means, Hierarchical Clustering) or similarity algorithm. Find the total number of applications requiring slice to efficiently orchestrate based on priority and availability. Identify applications that can share network slice and subslice for devices on the enterprise network as required. Use the machine learning classification model to classify features to orchestrate slice

Step 4. Understand trends for classification using – Explanatory Algorithms (SHAP or LIME)

Step 5. Define each slice and allocate to grouped applications

Step 6. Auto assign a network slice to new services with similar features/characteristics using - Similarity Algorithms (KNN, SVD, Cosine, Euclidean Distance, Cosine, Levenshtein, Jaro-Winkler, SVD etc…)

Step 7. Go back to the start to collect new data feed

The types of datasets used throughout this study are mentioned in Chapter 6.1. Each of these stages of the process was further discussed in the remainder of this Chapter, alongside their associated algorithms.

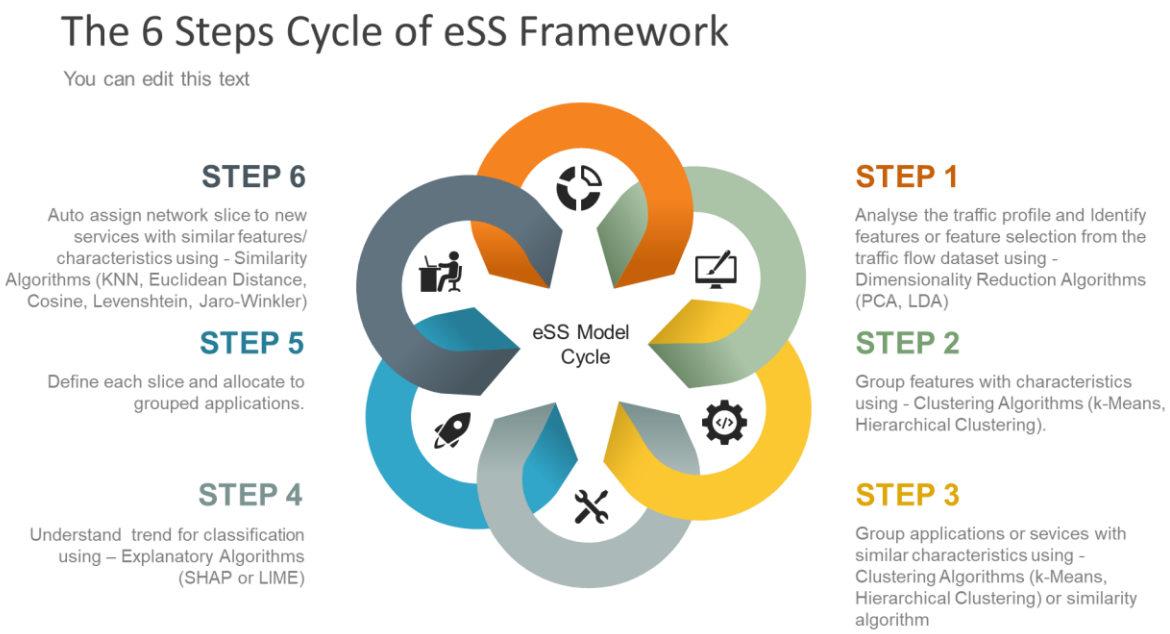


Figure 6.1.1: The 6 Steps Cycle of eSS Framework

Each stage is further discussed as follows from Chapter 6.1.1 to Chapter 6.1.6:

6.1.1 Step 1: Identify features or feature selection from the traffic flow dataset using - Dimensionality Reduction Algorithms (PCA, LDA)

The first stage was to select features from the collected data. This was done by first analysing the data. The analysed dataset was then passed through a series of machine learning algorithms to classify features. The result is then used to send requirements to orchestrate the network slice.

The collected network data is cleaned and pre-processed by eliminating irrelevant or inaccurate columns to increase the effectiveness of the machine learning model training. Rows with null, infinity or Not-a-Number (NaN) were also handled. The cleaning and preprocessing stage also included encoding the non-numerical columns using the sklearn library's LabelEncoder class. Categorical columns were also converted and handled.

Once the data had been cleaned, this was then passed on for feature engineering to identify the relevant columns. This is essential for the available dataset to be able to efficiently orchestrate the slice. This was done using Principal Component Analysis (PCA). The output was verified against AutoGluon machine learning algorithm which an extract of its result is shown in Figure 6.1.1.1.



Figure 6.1.1.1: Unused Features

The dimensional reduction algorithm that was used in this research to reduce the dimensionality of the dataset from 87 and identify the important features was Principal Component Analysis (PCA). PCA

was chosen to use in this research over the other available dimensionality reduction algorithm methods because the dataset being used has fewer zeros and therefore dense data unlike Singular Value Decomposition Algorithm which is better for datasets with many zeros or sparse data and Linear Discriminant Analysis Algorithm which has a limit of one number of the number of dimensions for projections [40]. Dimension reduction was employed to reduce unnecessary computational cost and improve the performance of the machine learning algorithms [44] used to formulate the proposed framework in this research. 33 important features were identified and selected out of the 87 features available in the original dataset by using the principal component analysis (PCA) method. Scree chart was then plotted to further reduce the selected features to the most important ones. This streamlined the principal components to 16 important features. This Step 1 in this cycle was carried out in three stages. Stage A involved the preparing, scaling and the dimensionality reduction of the dataset. The over 3 million rows in the dataset used in this research was also split into training and testing models fitted into PCA to be used for the research. Stage B involved the further analysis of the reduced dimension data to further streamline the identified number of principal components using scree plot. Stage C further analysed the streamlined principal components using biplot and boxplots to further validate findings and verify the resulting dataset which will be used for the research.

Stage A: Preparing, Scaling and Dimension reduction.

The below code and charts illustrate how the 87 features were reduced to 16 important ones in the dataset for analysis and network slicing requirements.

Firstly, the necessary libraries were imported as shown in Figure 6.1.1.1.

```
# import libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.ticker as mticker
%matplotlib inline
```

Figure 6.1.1.1: Import libraries.

The dataset was then loaded for analysis as shown in Figure 6.1.1.2.

```
#import data
data = pd.read_csv(r'C:\Users\abimbola.efunogbon\Downloads\archive\Dataset-Unicauca-Version2-87Atts.csv')
print (data.head())
data.iloc[:,1:].describe()
print (data.columns)
data.keys()
```

Figure 6.1.1.2: Load dataset

Figure 6.1.1.3 shows the data returned by the head syntax and Figure 6.1.1.4 shows the columns in the data returned. These are the features from the network traffic data profile collected.

```
                                Flow.ID      Source.IP  Source.Port  \
0    172.19.1.46-10.200.7.7-52422-3128-6    172.19.1.46        52422
1    172.19.1.46-10.200.7.7-52422-3128-6     10.200.7.7         3128
2   10.200.7.217-50.31.185.39-38848-80-6   50.31.185.39           80
3   10.200.7.217-50.31.185.39-38848-80-6   50.31.185.39           80
4  192.168.72.43-10.200.7.7-55961-3128-6  192.168.72.43        55961

  Destination.IP  Destination.Port  Protocol           Timestamp  \
0     10.200.7.7              3128         6  26/04/201711:11:17
1    172.19.1.46             52422         6  26/04/201711:11:17
2   10.200.7.217             38848         6  26/04/201711:11:17
3   10.200.7.217             38848         6  26/04/201711:11:17
4     10.200.7.7              3128         6  26/04/201711:11:17

   Flow.Duration  Total.Fwd.Packets  Total.Backward.Packets  ...  Active.Std  \
0          45523                 22                      55  ...         0.0
1              1                  2                       0  ...         0.0
2              1                  3                       0  ...         0.0
3            217                  1                       3  ...         0.0
4          78068                  5                       0  ...         0.0

   Active.Max  Active.Min  Idle.Mean  Idle.Std  Idle.Max  Idle.Min   Label  \
0         0.0         0.0        0.0       0.0       0.0       0.0  BENIGN
1         0.0         0.0        0.0       0.0       0.0       0.0  BENIGN
2         0.0         0.0        0.0       0.0       0.0       0.0  BENIGN
3         0.0         0.0        0.0       0.0       0.0       0.0  BENIGN
4         0.0         0.0        0.0       0.0       0.0       0.0  BENIGN
```

Figure 6.1.1.3: Top 5 data returned by the head ()

```
[5 rows x 87 columns]
Index(['Flow.ID', 'Source.IP', 'Source.Port', 'Destination.IP',
       'Destination.Port', 'Protocol', 'Timestamp', 'Flow.Duration',
       'Total.Fwd.Packets', 'Total.Backward.Packets',
       'Total.Length.of.Fwd.Packets', 'Total.Length.of.Bwd.Packets',
       'Fwd.Packet.Length.Max', 'Fwd.Packet.Length.Min',
       'Fwd.Packet.Length.Mean', 'Fwd.Packet.Length.Std',
       'Bwd.Packet.Length.Max', 'Bwd.Packet.Length.Min',
       'Bwd.Packet.Length.Mean', 'Bwd.Packet.Length.Std', 'Flow.Bytes.s',
       'Flow.Packets.s', 'Flow.IAT.Mean', 'Flow.IAT.Std', 'Flow.IAT.Max',
       'Flow.IAT.Min', 'Fwd.IAT.Total', 'Fwd.IAT.Mean', 'Fwd.IAT.Std',
       'Fwd.IAT.Max', 'Fwd.IAT.Min', 'Bwd.IAT.Total', 'Bwd.IAT.Mean',
       'Bwd.IAT.Std', 'Bwd.IAT.Max', 'Bwd.IAT.Min', 'Fwd.PSH.Flags',
       'Bwd.PSH.Flags', 'Fwd.URG.Flags', 'Bwd.URG.Flags', 'Fwd.Header.Length',
       'Bwd.Header.Length', 'Fwd.Packets.s', 'Bwd.Packets.s',
       'Min.Packet.Length', 'Max.Packet.Length', 'Packet.Length.Mean',
       'Packet.Length.Std', 'Packet.Length.Variance', 'FIN.Flag.Count',
       'SYN.Flag.Count', 'RST.Flag.Count', 'PSH.Flag.Count', 'ACK.Flag.Count',
       'URG.Flag.Count', 'CWE.Flag.Count', 'ECE.Flag.Count', 'Down.Up.Ratio',
       'Average.Packet.Size', 'Avg.Fwd.Segment.Size', 'Avg.Bwd.Segment.Size',
       'Fwd.Header.Length.1', 'Fwd.Avg.Bytes.Bulk', 'Fwd.Avg.Packets.Bulk',
       'Fwd.Avg.Bulk.Rate', 'Bwd.Avg.Bytes.Bulk', 'Bwd.Avg.Packets.Bulk',
       'Bwd.Avg.Bulk.Rate', 'Subflow.Fwd.Packets', 'Subflow.Fwd.Bytes',
       'Subflow.Bwd.Packets', 'Subflow.Bwd.Bytes', 'Init_Win_bytes_forward',
       'Init_Win_bytes_backward', 'act_data_pkt_fwd', 'min_seg_size_forward',
       'Active.Mean', 'Active.Std', 'Active.Max', 'Active.Min', 'Idle.Mean',
       'Idle.Std', 'Idle.Max', 'Idle.Min', 'Label', 'L7Protocol',
       'ProtocolName'],
```

Figure 6.1.1.4: All data features available in dataset

The loaded dataset contained some non-numerical numbers which will make the data not easily plottable for analysis. The dataset was converted to numerical values for non-numerical values as shown in Figure 6.1.1.5. In this research, the Flow.ID, Source.IP, Destination.IP, Label, ProtocolName and Timestamp from the dataset were converted to numerical values.

```
:  #make numerical

   from sklearn.preprocessing import LabelEncoder

   le = LabelEncoder()
   le.fit(data['Flow.ID'])

   # apply encoding to labels
   data['Flow.ID'] = le.transform(data['Flow.ID'])
   le.fit(data['Source.IP'])
   data['Source.IP'] = le.transform(data['Source.IP'])
   le.fit(data['Destination.IP'])
   data['Destination.IP'] = le.transform(data['Destination.IP'])
   le.fit(data['Label'])
   data['Label'] = le.transform(data['Label'])
   le.fit(data['ProtocolName'])
   data['ProtocolName'] = le.transform(data['ProtocolName'])
   le.fit(data['Timestamp'])
   data['Timestamp'] = le.transform(data['Timestamp'])

   print (data.head())
```

Figure 6.1.1.5: Make data numerical.

The converted data is shown using the head syntax which displays the top 5 records.

```
print (data.head())
```

```
   Flow.ID  Source.IP  Source.Port  Destination.IP  Destination.Port  \
0   430798       1291        52422              65              3128
1   430798         62         3128            5197             52422
2   224539       4697           80              59             38848
3   224539       4697           80              59             38848
4  1270645       3181        55961              65              3128

   Protocol  Timestamp  Flow.Duration  Total.Fwd.Packets  \
0         6      22694          45523                 22
1         6      22694              1                  2
2         6      22694              1                  3
3         6      22694            217                  1
4         6      22694          78068                  5

   Total.Backward.Packets  ...  Active.Std  Active.Max  Active.Min  Idle.Mean  \
0                      55  ...         0.0         0.0         0.0        0.0
1                       0  ...         0.0         0.0         0.0        0.0
2                       0  ...         0.0         0.0         0.0        0.0
3                       3  ...         0.0         0.0         0.0        0.0
4                       0  ...         0.0         0.0         0.0        0.0

   Idle.Std  Idle.Max  Idle.Min  Label  L7Protocol  ProtocolName
0       0.0       0.0       0.0      0         131            28
1       0.0       0.0       0.0      0         131            28
2       0.0       0.0       0.0      0           7            25
3       0.0       0.0       0.0      0           7            25
4       0.0       0.0       0.0      0         131            28
```

Figure 6.1.1.6: Converted Dataset into Numerical

The dataset was then normalised for PCA in order for all the features to have a mean of 0 and standard deviation of 1. The applications running that need the network slices whose features are being measured in terms of packets are the ProtocolName. This was dropped from the X axis data as it will be used as reference point for Y axis.

```
#Normalize the data for PCA

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X= data.drop('ProtocolName', axis=1)
y=data['ProtocolName']
```

Figure 6.1.1.7: Import libraries for scaling data and separating the application reference point.

The dataset was then split into a set of data that can be used for training and another set for testing and validating the model.

```
#split data
from sklearn.model_selection import train_test_split
train_img, test_img, train_lbl, test_lbl = train_test_split( data.drop('ProtocolName', axis=1), data['ProtocolName'], test_size=1
```

Figure 6.1.1.8: Data split for training and testing model

The training data set was then scaled and fitted.

```
scaler.fit(train_img)

# Apply transform to both the training set and the test set.
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```

Figure 6.1.1.9: Fitted training model and transformed dataset.

The data was split in the ratio of 1/7. The resulting number of records in the scaled and splitted data of over 3 million rows can be shown with the code below. The training model has over 3 million records to train and the testing model has 511,043 records.

The output is shown below in Figure 6.1.1.10.

```
print("train_img= ",len(train_img))
print("test_img= ",len(test_img))
```

```
train_img=  3066253
test_img=  511043
```

Figure 6.1.1.10: Number of records in training and testing model

After cleaning and preparing the data, PCA was then applied on the dataset.

```
from sklearn import preprocessing

# Import the PCA module
from sklearn.decomposition import PCA
pca=PCA(n_components=.95)

#Fitting PCA on training set only.
dfx_pca= pca.fit(train_img)
train_img = pd.DataFrame(preprocessing.scale(train_img),columns = data.drop('ProtocolName', axis=1).columns)
```

Figure 6.1.1.11: PCA application on the training model

Figure 6.1.1.11 shows 95% of the component's variance was used and the training model fitted into PCA. The feature sets were automatically reduced to 33 by PCA as the important features out of the 87 initial features. PCA is used in this study to reduce dimensionality, reduce computational cost, and increase accuracy [54]. The following code was used to print and confirm the number of features identified by PCA as the important features.

>> print("n_components_", pca.n_components_)

The resulting number of components and the resulting shape of the training model is shown in the output on Figure 6.1.1.12.

```
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
```

```
print("Shape of training model = ",train_img.shape)
print("Number of important features reduced by PCA = ", pca.n_components_)
```

```
Shape of training model =  (3066253, 33)
Number of important features reduced by PCA =  33
```

Figure 6.1.1.12: Output showing the number of features.

In Figure 6.1.1.12, the first couple of lines of code transformed the training and test model with PCA and the following line of code shows the number of resulting rows and columns as well as the total count of the number of columns or features selected by PCA.

PCA has identified 33 components as the important features and the names of these features as shown in Figure 6.1.1.14:

These feature names were extracted using the following code in Figure 6.1.1.13.

```
most_important = [np.abs(pca.components_[i]).argmax() for i in range(pca.n_components_)]

# get the names
most_important_names = [data.columns[most_important[i]] for i in range(pca.n_components_)]

list_feat = {'PC{}'.format(i): most_important_names[i] for i in range(pca.n_components_)}

# build dataframe
df = pd.DataFrame(list_feat.items())

string_features = (df
                   .select_dtypes(include='object')
                   .columns.tolist())

print("selected_feat = ",df)
```

Figure 6.1.1.13: Extract important feature names.

The resulting output is shown in Figure 6.1.1.14.

0	PC0	Idle.Max
1	PC1	Max.Packet.Length
2	PC2	Subflow.Fwd.Packets
3	PC3	Fwd.Packet.Length.Mean
4	PC4	PSH.Flag.Count
5	PC5	Active.Mean
6	PC6	Destination.IP
7	PC7	Total.Length.of.Bwd.Packets
8	PC8	Fwd.IAT.Min
9	PC9	Flow.Packets.s
10	PC10	RST.Flag.Count
11	PC11	L7Protocol
12	PC12	Bwd.IAT.Min
13	PC13	Min.Packet.Length
14	PC14	Flow.IAT.Min
15	PC15	Init_Win_bytes_backward
16	PC16	Flow.IAT.Min
17	PC17	Timestamp
18	PC18	Protocol
19	PC19	Timestamp
20	PC20	Bwd.Packet.Length.Min
21	PC21	Flow.Bytes.s
22	PC22	Idle.Std
23	PC23	Flow.ID
24	PC24	Down.Up.Ratio
25	PC25	Fwd.IAT.Std
26	PC26	Bwd.Packets.s
27	PC27	Flow.ID
28	PC28	Bwd.IAT.Min
29	PC29	L7Protocol
30	PC30	Min.Packet.Length
31	PC31	Total.Length.of.Fwd.Packets
32	PC32	Bwd.IAT.Std

Figure 6.1.1.14: Selected 33 Principal Components

A histogram chart was plotted for these extracted features to further understand the distribution of data as shown in Figure 6.1.1.15.

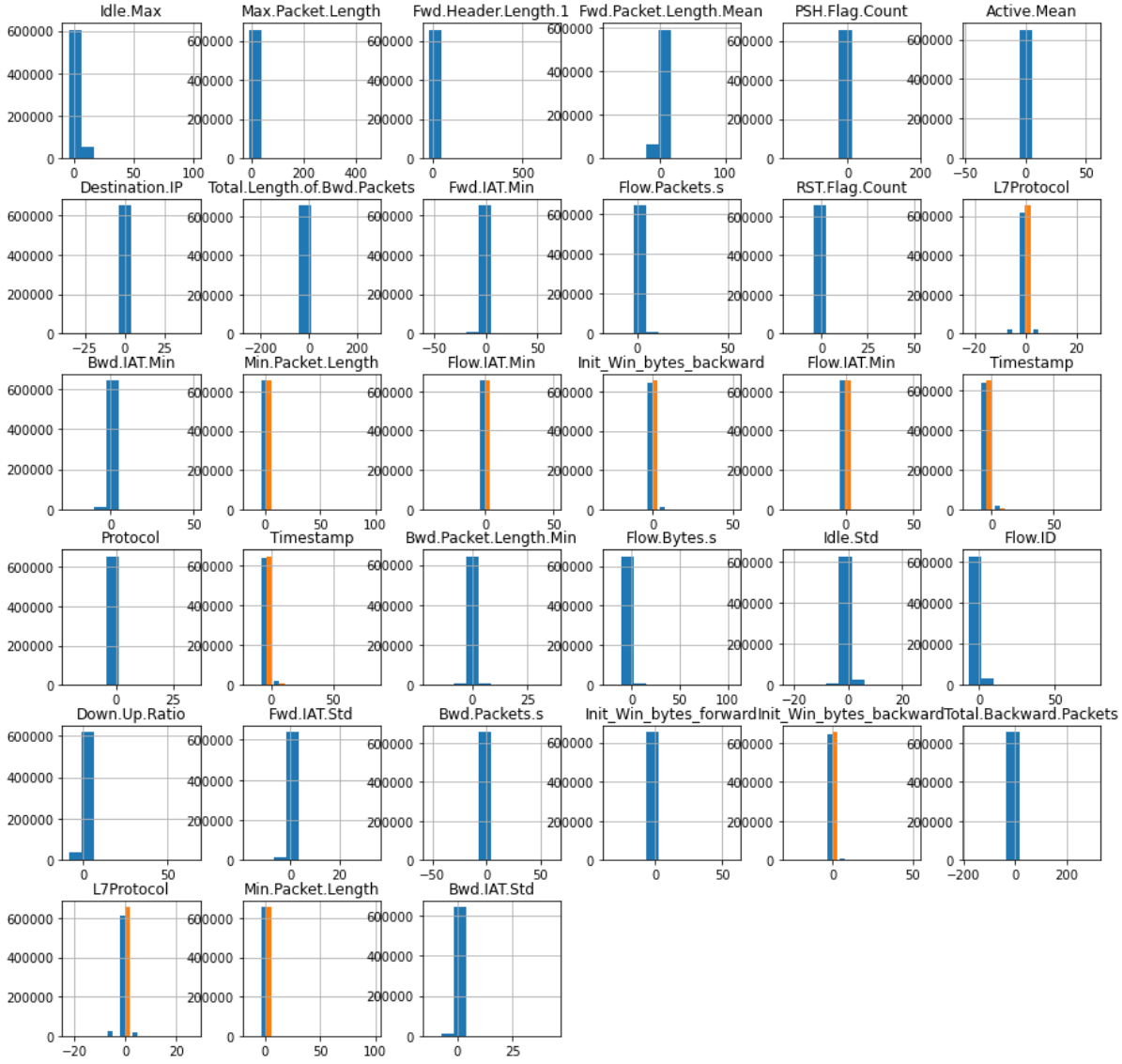


Figure 6.1.1.15: Histogram of selected features

With 33 principal components forming multidimensional data, further analysis is carried out to further streamline the number of principal components to the most important ones. This was done by plotting the scree chart and analysing the result. The number of features was further reduced to 16 which are then used throughout the research for analysis. Stage B of this step demonstrates the scree plot and the analysis.

Stage B: Streamlining the Principal Components

A scree plot was plotted with the percentage variance against number of features and 80% accuracy. The number of reduced principal components to describe 80% of the variance is 16 as shown in Figure 6.1.1.16.

```
screeplot =np.cumsum( np.round(pca.explained_variance_ratio_, decimals=3)*100)
plt.ylabel('% variance')
plt.xlabel('# of features')
plt.title('scree plot')
plt.ylim(min(screeplot),100.5)
plt.style.context('seaborn-whitegrid')
plt.axhline(y=80, color='r', linestyle ='--')
plt.plot(screeplot)
plt.show
```

Figure 6.1.1.16: Code for scree plot

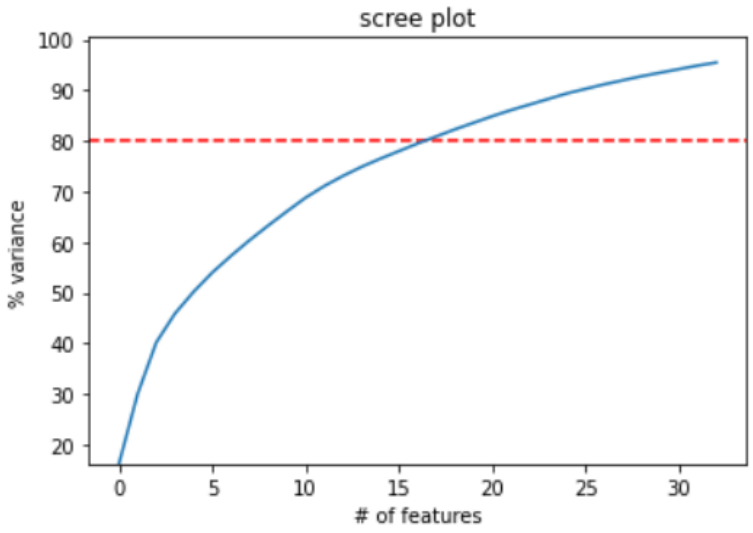


Figure 6.1.1.17: Scree Plot

Eigenvalues were also plotted to validate the number of reduced principal components and the number of principal components to have a value of 1 or higher was also shown as 16. Figure 6.1.1.18 shows the code used to plot the chat and Figure 6.1.1.19 shows the generated graph.

```
plt.ylabel('Eigenvalues')
plt.xlabel('# of features)')
plt.title('Eigenvalues')
plt.ylim(0,max(pca.explained_variance_))
plt.style.context('seaborn-whitegrid')
plt.axhline(y=1, color='r', linestyle ='--')
plt.plot(pca.explained_variance_)
plt.show
```

Figure 6.1.1.18: Plotting the Eigenvalues against the number of features.

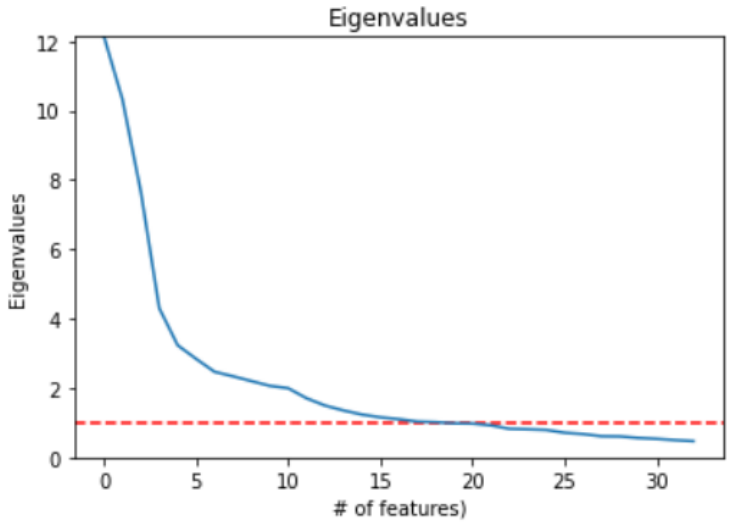


Figure 6.1.1.19: Scree plot of Eigenvalues against the number of features

Using the eigenvalues as shown above, the elbow can be seen at PC10, and PC16 shows that value just before the curve flattens out. It can be deduced from the plot above that up to PC16 has eigenvalues greater than or equal to 1 and has the elbow point just before the curve flattens out. This also follows Kaiser's rule [41]. Further analysis and validation of the number of selected principal components was further carried out by plotting the explained variance ratio against the principal components as shown in Figure 6.1.1.20.

```
PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_, 'ro-', linewidth=2)
plt.title('Variance Ratio against Principal Components')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.show()
```

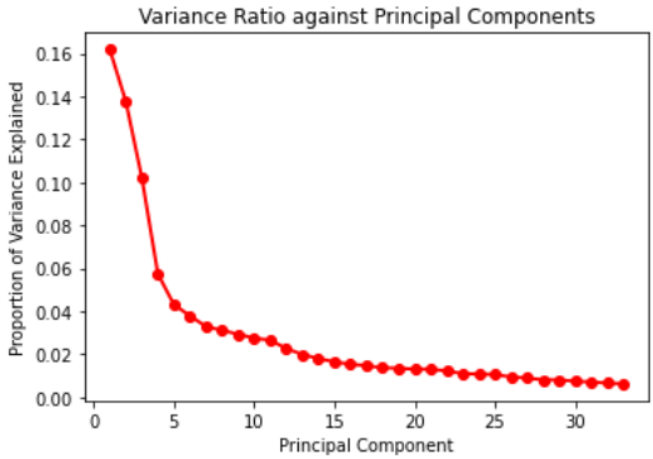


Figure 6.1.1.20: Explained variance ratio against principal components

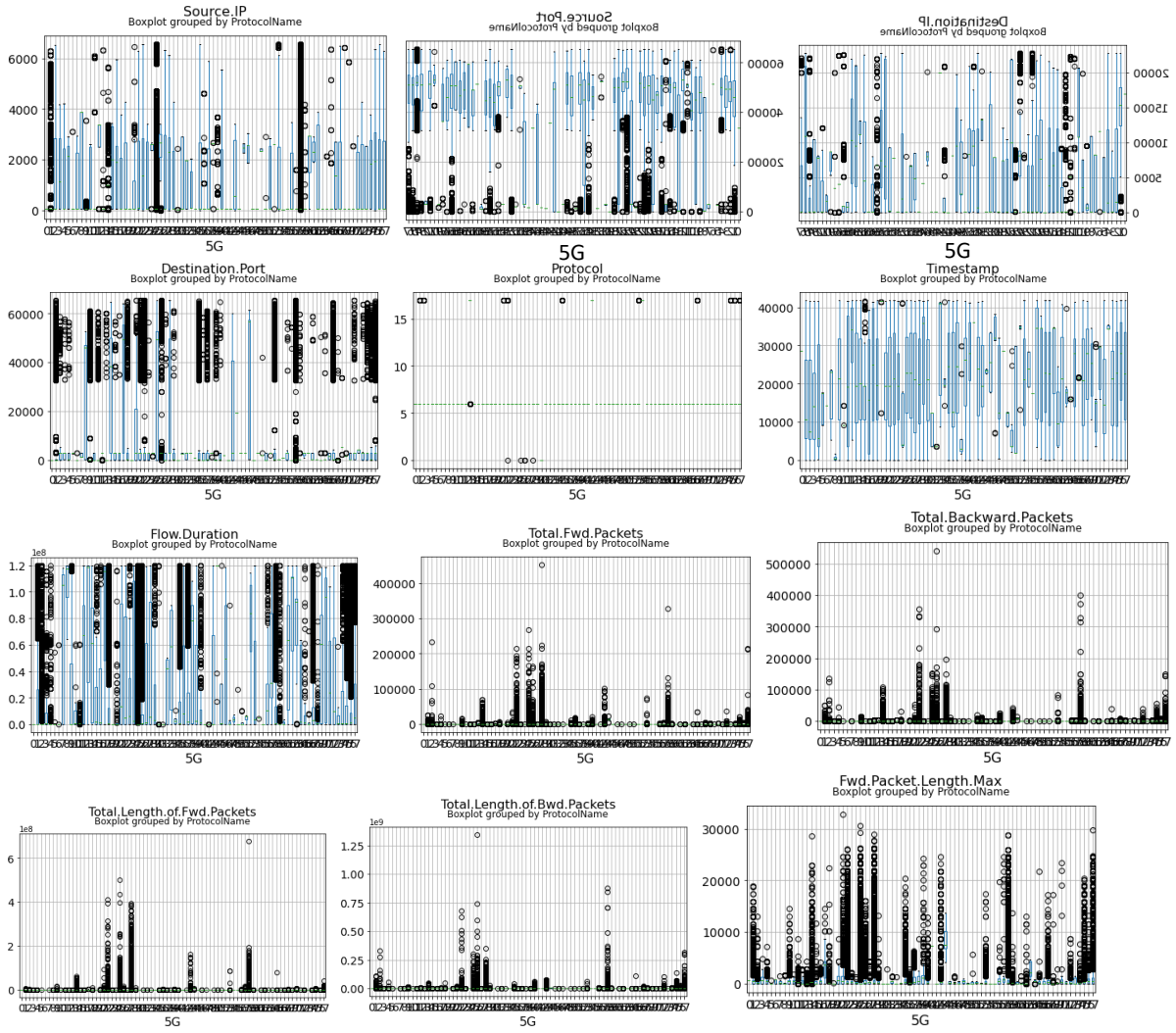
At the end of Stage B of Step 1, 33 principal components have been reduced to 16 for use in the machine learning models later on in this research to automatically orchestrate the slice based on the features request.

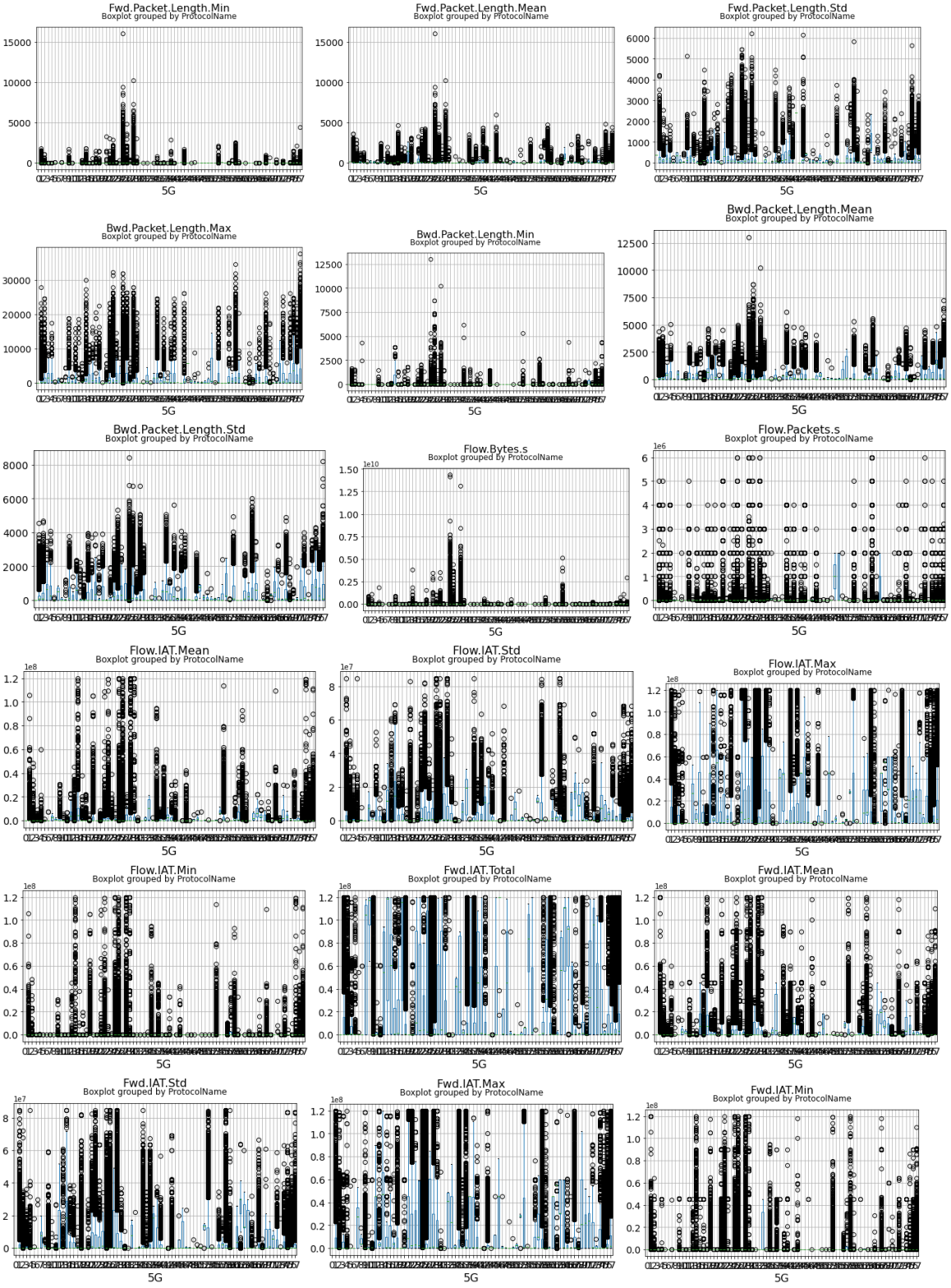
Stage C: Validating findings so far

The final stage of Step 1 is to validate the results of this step by plotting boxplots and biplots to visualise the data and validate the reduced number of features.

The boxplot which was plotted for each feature by the apllication which is the protocol name visually showed features that were less important and those that could be analysed similarly with the same trend of data. The code and sample result is shown in Figure 6.1.1.21. This shows 33 of 87 important features and about 16 of which could be easily distinguished.

```
for c in data.columns[1:]:
    data.boxplot(c,by='ProtocolName',figsize =(7,4) , fontsize = 14)
    plt.title("{}\n".format(c),fontsize=16)
    plt.xlabel("5G", fontsize=16)
```




Fwd.Packet.Length.Min
Boxplot grouped by ProtocolName
Fwd.Packet.Length.Mean
Boxplot grouped by ProtocolName
Fwd.Packet.Length.Std
Boxplot grouped by ProtocolName
Bwd.Packet.Length.Max
Boxplot grouped by ProtocolName
Bwd.Packet.Length.Min
Boxplot grouped by ProtocolName
Bwd.Packet.Length.Mean
Boxplot grouped by ProtocolName
Bwd.Packet.Length.Std
Boxplot grouped by ProtocolName
Flow.Bytes.s
Boxplot grouped by ProtocolName
Flow.Packets.s
Boxplot grouped by ProtocolName
Flow.IAT.Mean
Boxplot grouped by ProtocolName
Flow.IAT.Std
Boxplot grouped by ProtocolName
Flow.IAT.Max
Boxplot grouped by ProtocolName
Flow.IAT.Min
Boxplot grouped by ProtocolName
Fwd.IAT.Total
Boxplot grouped by ProtocolName
Fwd.IAT.Mean
Boxplot grouped by ProtocolName
Fwd.IAT.Std
Boxplot grouped by ProtocolName
Fwd.IAT.Max
Boxplot grouped by ProtocolName
Fwd.IAT.Min
Boxplot grouped by ProtocolName
5G

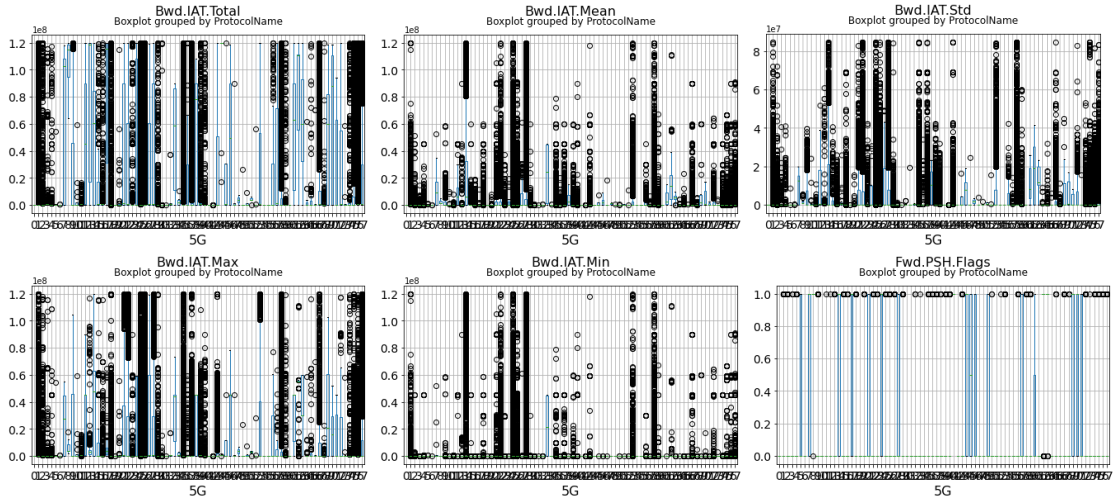


Figure 6.1.1.21: Boxplot of all features against ProtocolName

With the boxplot validating the findings, the correlation matrix was plotted to visually demonstrate the correlated features as shown in Figure 6.1.1.22. Two features were also plotted to visualise on a scatter plot as shown in Figure 6.1.1.23 and two principal components plotted and data visualised to see how the data relate as shown in Figure 6.1.1.24.

```
# Plot the correlation matrix
def correlation_matrix(data):
    from matplotlib import pyplot as plt
    from matplotlib import cm as cm

    label_format = '{:}'

    fig = plt.figure(figsize=(16,12))
    ax1= fig.add_subplot(111)
    cmap=cm.get_cmap('jet',30)
    cax= ax1.imshow(data.corr(), interpolation="nearest", cmap=cmap)
    ax1.grid(True)
    plt.title('features correlation\n', fontsize=15)
    labels =data.columns
    ax1.set_xticklabels(labels,fontsize=12)
    ax1.set_yticklabels(labels,fontsize=12)
    fig.colorbar(cax,ticks=[0.1*i for i in range (-11,11)])
    plt.show()


correlation_matrix(data)
```

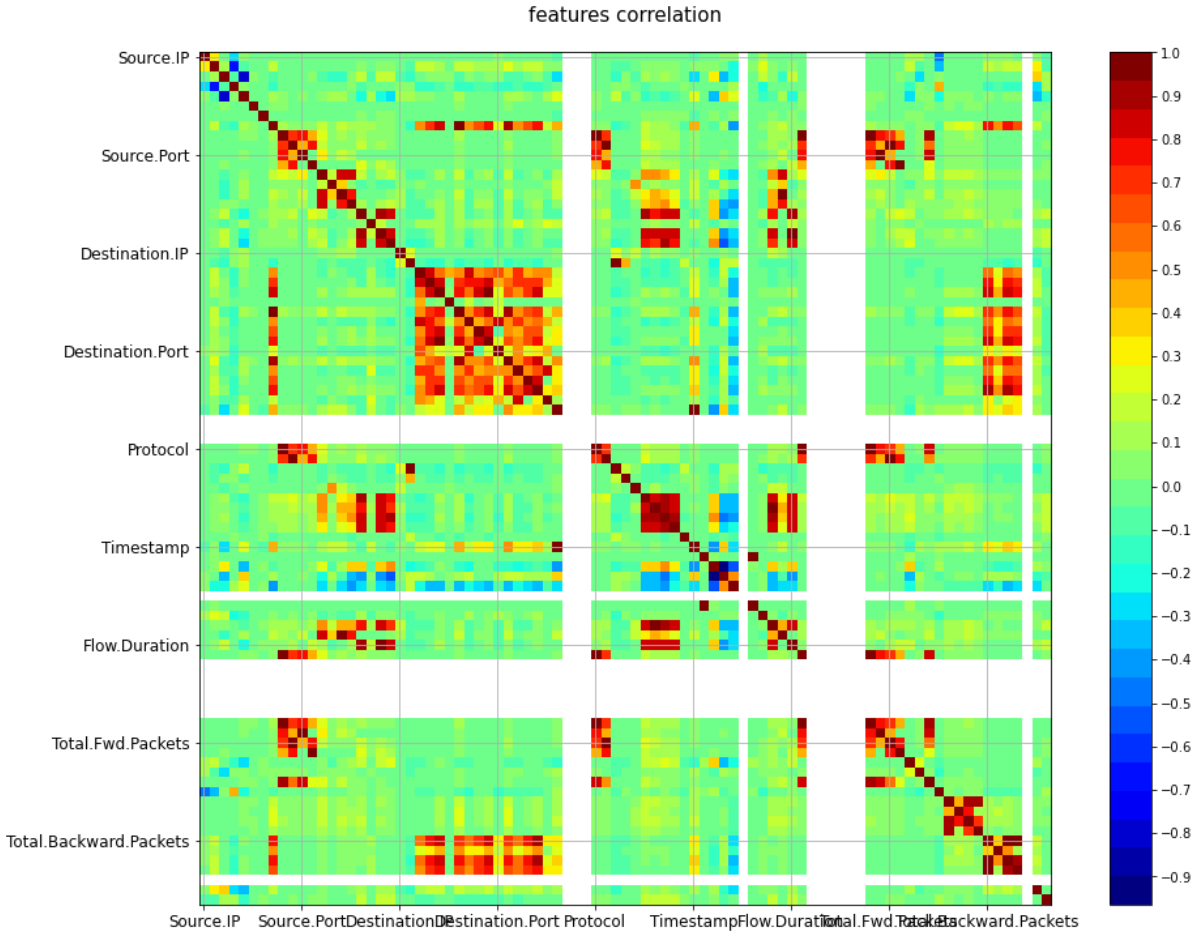


Figure 6.1.1.22: Correlation matrix

```
#plot 2 variables
plt.figure(figsize=(10,6))
plt.scatter(data['Max.Packet.Length'],data['Subflow.Fwd.Packets'],c=data['ProtocolName'], edgecolors='k', alpha=0.8, cmap=plt.cm.
plt.grid(True)
plt.title("scatter plot \n 2 variables",fontsize=15)
plt.xlabel('Max.Packet.Length')
plt.ylabel('ProtocolName')
```

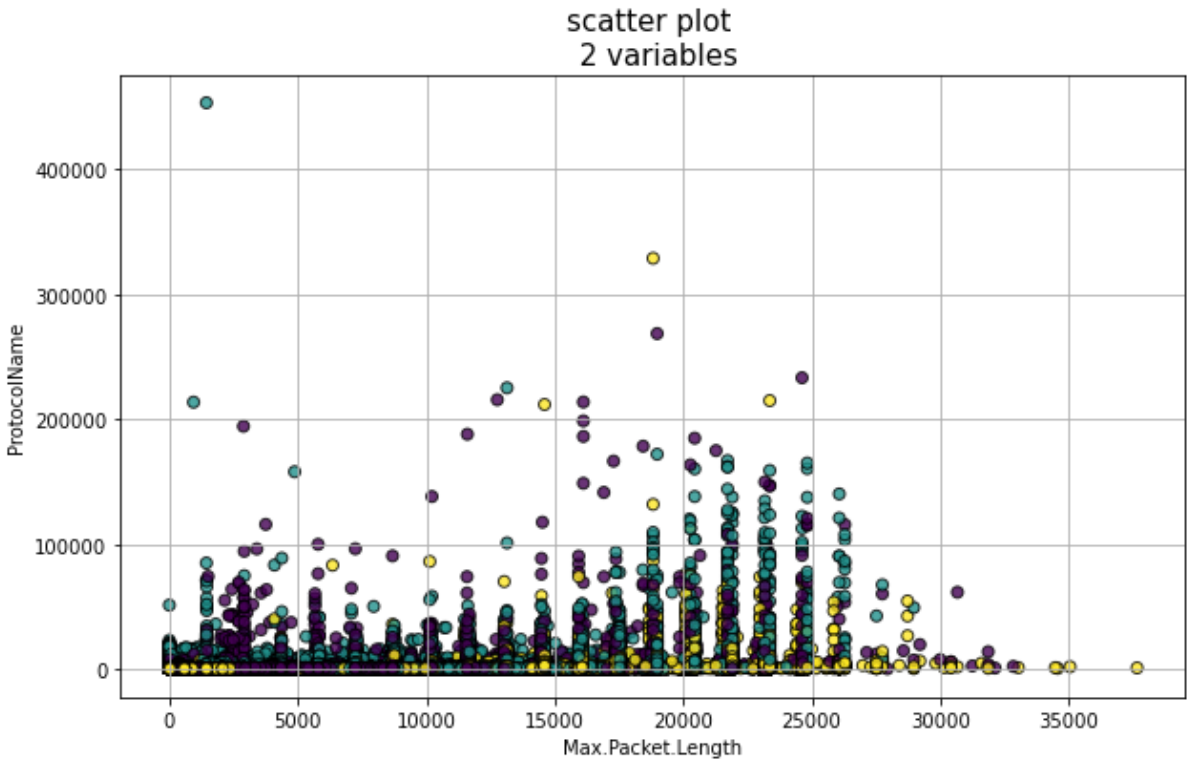


Figure 6.1.1.23: Scatter plot of two features

```
plt.figure(figsize=(8,6))
plt.scatter(data['Idle.Max'],data['Max.Packet.Length'],c=data['ProtocolName'],edgecolors='m', s=150,alpha=0.75)
plt.xlabel('First principle component')
plt.ylabel('Second principle component')
plt.show()
```

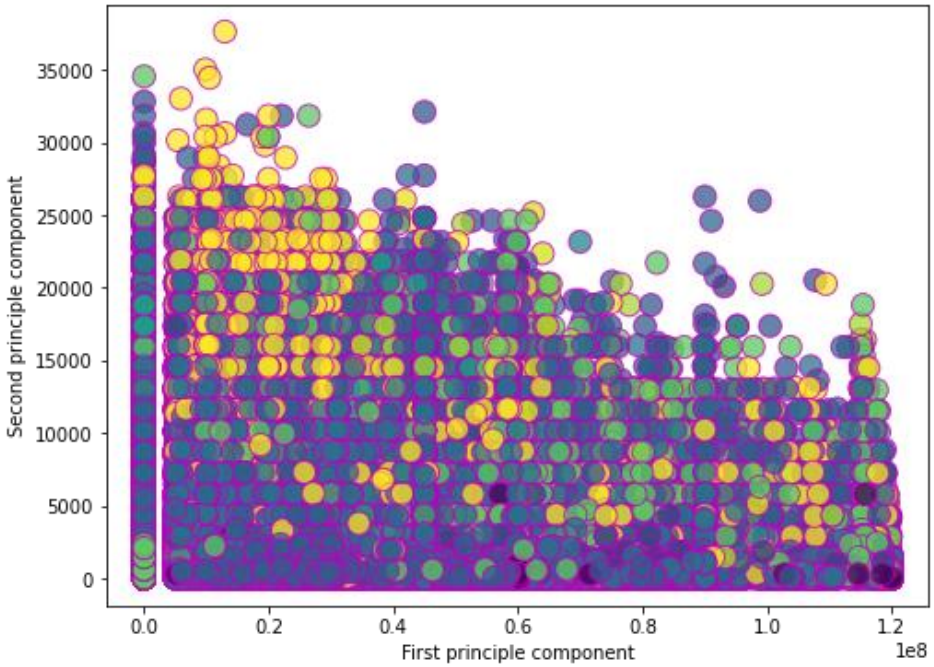


Figure 6.1.1.24: PC1 against PC2

The network slice orchestration and sub slicing go through the design and planning phase, the construction and initiation stage, the management and monitoring phase as well as the deployment and revoke phase. The layer at each phase is shown in Figure 4.1 with the respective ML algorithm proposed shown. The study shows given the network data available, the proposed models outperform others with high accuracy and speed. Figure 4.1 shows the input, process, and predicted output of the proposed CNaaS framework.

6.1.2 Step 2: Group features with characteristics using – LazyPredict Algorithms

The classification of features will be automated and will be using the statistical classification and deep packet inspection principle to group the various services and determine the characteristics of network slice required [43]. The packet data collected from the network per application is analysed and used for formulating part of the network slice characteristics that is orchestrated on demand. With the amount of data generated and the rate at which new applications are made to meet users need and demand 5G network slices, unsupervised machine learning algorithm has been selected to use for clustering the data into the similar network characteristics which are the network slices. An unsupervised learning machine learning algorithm was chosen because it is not aware of the output and finds correlation between the input to group similar characteristics. This will help with re-orchestration of network slices to meet new demand on fly should the application service network slice demand changes. K-means is used to determine the number of grouped subslice to be orchestrated based on similar characteristics and applications while the LazyPredict is used to determine the machine learning algorithm most suitable for the available dataset to extract features and subslice requirements, classify the applications and devices on the network and determining which network function services can be shared based on the network demand and requirements with efficiently allocating them to appropriate subslice template group. The random Forest algorithm as shown in Figure 6.1.2.2 was chosen due to its fast and scalable ability.

Figure 6.1.2.1 shows the elbow point for determining the number of Ks which is the number of clusters in the data. K-means was also used due to its fast and scalable ability just behind random forest. A combination of the two best models were used for training and analysis. The elbow to determine K can be seen at point 6.

```
from sklearn.cluster import KMeans

sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(train_img)
    sse.append(km.inertia_)

plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

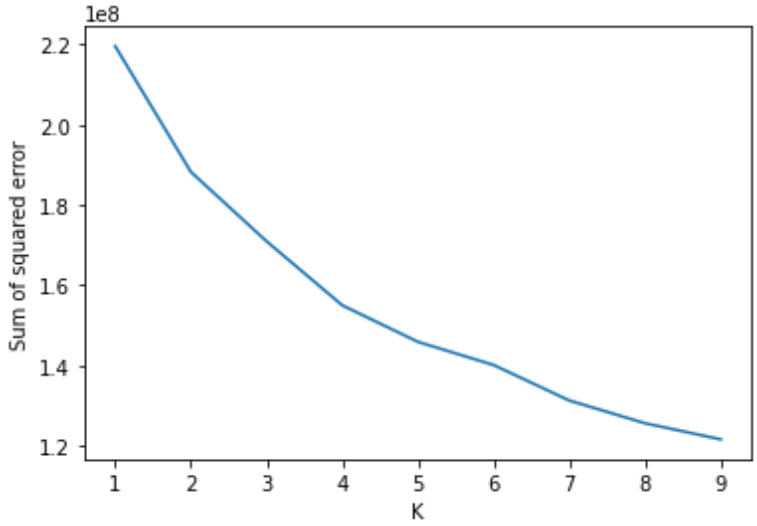


Figure 6.1.2.1: K-Means clustering.

6.1.2.1 Applications classification

Multiple devices may run the same type of applications or a mix of same applications and will require the specific orchestrated slice to be shared by the same type of applications on the various devices. The applications may run on static or dynamic ports. Machine learning is used to develop the model to overcome the existing challenge of classifying dynamic ports such as encrypted ports, secured ports or virtual private network ports [60] due to the changing of network port unlike static ports such as HTTP, SMTP, FTP, SSH and other static network ports [59]. The machine model will also be able to classify other types of applications running on dynamic network port and able to classify any new application that comes on the network. Five classification machine learning algorithms are studied and compared in this research to classify the application services on the network to the group of features for network slice: K-nearest Neighbors (K-NN), Decision tree, Support vector machine (SVM), Random Forest and Artificial neural network (ANN)[61].

K-NN uses pattern recognition to group application services to its nearest k relative neighbour. Application services will for example be grouped in a class nearest 1 if k=1.

Decision tree classifies application services by ordering or forming subgroups of groups. Using the structure of a tree such as subgrouping into branches, and then subgrouping into leaves until the similarities between the features are very close. This model involves two main steps namely induction and pruning where the former builds the tree, and the latter removes all unnecessary complexities from the structure.

Support vector machine (SVM) classifies application service features based on polarity using X/Y prediction. The hyperplane that differentiates the application services into the best class is then calculated.

Random Forest has a complex algorithm with being a layer on decision tree. It can perform classification and regression of data input as well as other tasks.

Artificial neural network algorithm simulates the biological brain and how neurons are interlinked in the brain to understand data and classify applications service features.

```
In [16]: import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('SVM', SVC()))
```

```
In [18]: results = []
names = []
datamla = []
scoring = 'accuracy'
for name, model in models:
        kfold = model_selection.KFold(n_splits=5)
        cv_results = model_selection.cross_val_score(model, train_img2, train_lbl2, cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
        datamla.append([name, cv_results.mean(), cv_results.std()])

       #dfmla = pd.DataFrame(datamla, columns=['MLA used', 'Train Accuracy', 'Test Accuracy','Precission', 'Recall', 'AUC'])


dfmla = pd.DataFrame(datamla, columns=['MLA used', 'Mean', 'Standard Deviation'])
```

```
LR: 0.780266 (0.005731)
LDA: 0.746747 (0.002378)
KNN: 0.825998 (0.000625)
CART: 0.815137 (0.000572)
NB: 0.262028 (0.009764)
RF: 0.880754 (0.000286)
  MLA used      Mean  Standard Deviation
0       LR  0.780266            0.005731
1      LDA  0.746747            0.002378
2      KNN  0.825998            0.000625
3     CART  0.815137            0.000572
4       NB  0.262028            0.009764
5       RF  0.880754            0.000286
```

Figure 6.1.2.2: Results after comparing various machine learning models' performance with training the dataset

```
from lazypredict.Supervised import LazyClassifier, LazyRegressor
classifiers = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)

models,predictions = classifiers.fit(train_img,test_img, train_lbl, test_lbl )
```

The printed model from the code above generated the following results showing Random Forest algorithm with highest accuracy and good speed.

Table 6.1.2.1: Results from LazyPredict module showing machine learning models performance with training the dataset.

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
:----	----------:	-------:	----------:	----------:	----------:
LinearSVC	0.991892	0.988969	0.989057	0.992723	0.0171418
SGDClassifier	0.991892	0.988969	0.989057	0.992723	0.0131402
MLPClassifier	0.988383	0.988329	0.988417	0.989255	0.428141
Perceptron	0.988383	0.986222	0.98631	0.989226	0.0141456
LogisticRegression	0.988383	0.984115	0.984203	0.989195	0.0221446
LogisticRegressionCV	0.988383	0.984115	0.984203	0.989195	0.265138
SVC	0.984874	0.981367	0.981455	0.985698	0.0161421
CalibratedClassifierCV	0.984874	0.977153	0.977241	0.985618	0.0371425
PassiveAggressiveClassifier	0.977857	0.975873	0.975961	0.978725	0.0151415
LabelPropagation	0.977857	0.975873	0.975961	0.978725	0.0451398
LabelSpreading	0.977857	0.975873	0.975961	0.978725	0.0331416
RandomForestClassifier	0.994348	0.991019	0.991107	0.985191	0.015141
GradientBoostingClassifier	0.974348	0.968911	0.968999	0.97513	0.169139
QuadraticDiscriminantAnalysis	0.96733	0.967631	0.967719	0.968313	0.0141404
HistGradientBoostingClassifier	0.970839	0.966164	0.966252	0.971648	0.684144
RidgeClassifierCV	0.974348	0.964697	0.964785	0.974997	0.0151439
RidgeClassifier	0.970839	0.96195	0.962038	0.971503	0.0141387
AdaBoostClassifier	0.963822	0.96067	0.960758	0.964705	0.207139
ExtraTreesClassifier	0.963822	0.958563	0.958651	0.964623	0.0291476
KNeighborsClassifier	0.963822	0.956455	0.956543	0.964537	0.0581415
BaggingClassifier	0.949786	0.956002	0.95609	0.951143	0.0581381
BernoulliNB	0.953295	0.952428	0.952516	0.954333	0.0191398
LinearDiscriminantAnalysis	0.963822	0.952241	0.952329	0.96435	0.0221405
GaussianNB	0.956804	0.950961	0.951049	0.957598	0.0161345
NuSVC	0.956804	0.94464	0.944728	0.957275	0.02213
DecisionTreeClassifier	0.93926	0.935118	0.935206	0.940232	0.0191433
NearestCentroid	0.949786	0.934931	0.935019	0.950062	0.0181484
ExtraTreeClassifier	0.925225	0.913593	0.913681	0.925723	0.0131409
CheckingClassifier	0.363822	0.501425	0.501513	0.19514	0.0191453
DummyClassifier	0.514699	0.491023	0.491111	0.522185	0.0141375

6.1.2.2 Optimise slice admission policy.

After determining the number of K and slices to be subslice d, the data was further analysed to prevent or reduce the possibilities of slice denial. Predicting the network slice characteristics and requirements reduces the possibility of slice denial and optimises the slice admission policy. Mobile network operators can forecast and plan for the required physical resources to avoid denial of service requests

and increase availability as well as reliability. The risk of breaching service level agreements (SLA) to meet the required quality of service needs is thereby reduced.

The total number of applications service types running on the enterprise network requiring a slice of the network is determined to help with forecasting and planning of required physical resources. This will help to prioritise network resources, understand the demand versus availability and efficiently share the available network resources amongst the application services. Figure 6.1.2.3 shows the code and result on our sample company dataset with 74 unique applications service types running on the enterprise network.

```
count_class = train_img3.groupby('ProtocolName').size()
print(count_class)
```

```
ProtocolName
99TAXI                1
AMAZON            37237
APPLE              3309
APPLE_ICLOUD        529
APPLE_ITUNES        541
                   ...
WHOIS_DAS             6
WIKIPEDIA           892
WINDOWS_UPDATE    14941
YAHOO              9206
YOUTUBE           73167
Length: 74, dtype: int64
```

Figure 6.1.2.3: Number of active application instances on company's network

6.1.2.3 Validating Result

After clustering to determine number of slices to be subsliced and determining the applications on the network that require slices with the characteristics required. The classification machine learning model is applied to the data to determine applications which require similar characteristics or can share network functions to optimise the available network slice and subslice d. Step 2 of this process of clustering and determining number of applications to share slices or own its own slice was then validated by running each model against the data and determining the accuracy, precision and fall back. The capability matrix was also determined. The result verified that Random Forest model was the model with the highest accuracy. Figure 6.1.2.4 shows the result of Random Forest with test 91% accuracy and 99% accuracy for train. Figure 6.1.2.5 shows the recall and precision score of 87%. Figure 6.1.2.6 shows the accuracy of Decision tree classifier as 86% with recall and precision score of 86%. Figure 6.1.2.8 shows the accuracy of K-NN as 89% with precision and recall score of 88% and 89% respectively. Figure 6.1.2.10 shows the classification report using K-NN on the dataset. The result shows the precision, recall, f1 score and support. This is compared with the Random Forest classification matrix shown in Figure 6.1.2.11. The scores are further verified with the result of the confusion matrix as shown in 6.1.2.13.

```
# import
from sklearn.ensemble import RandomForestClassifier

# initialize
clf = RandomForestClassifier()

# train the classifier using the training data
clf.fit(train_img, train_lbl)

# compute accuracy using test data
acc_test = clf.score(test_img, test_lbl)

print ("Test Accuracy:", acc_test)
```

```
Test Accuracy: 0.9105750396737652
```

```
# compute accuracy using training data
acc_train = clf.score(train_img, train_lbl)

print ("Train Accuracy:", acc_train)
```

```
# compute accuracy using training data
acc_train = clf.score(train_img2, train_lbl2)

print ("Train Accuracy:", acc_train)
```

```
Train Accuracy: 0.9999969561101523
```

Train Accuracy: 0.9999969561101523

Figure 6.1.2.4: Validating Models – Random Forest

```
from sklearn.metrics import recall_score, precision_score
```

```
pred = clf.predict(test_img2)

precision = precision_score(test_lbl2, pred, average="weighted")
recall = recall_score(test_lbl2, pred, average="weighted")
```

```
print ("Precision:", precision)
print ("Recall:", recall)
```

```
Precision: 0.8733848230181919
Recall: 0.8736896054205591
```

Precision: 0.8733848230181919

Recall: 0.8736896054205591

Figure 6.1.2.5: Precision and Recall Score – Random Forest

```
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
d=DecisionTreeClassifier()

d.fit(x2,train_lbl2)
```

```
DecisionTreeClassifier()
```

```
from sklearn.metrics import accuracy_score

result=d.predict(test_img2)
print(accuracy_score(test_lbl2,result))
```

```
0.8625638075409328
```

Figure 6.1.2.6: Validating Models - Decision Tree Classifier Model

```
from sklearn.metrics import f1_score, precision_score, recall_score, classification_report, confusion_matrix

precision = precision_score(test_lbl2, result, average="weighted")
recall = recall_score(test_lbl2, result, average="weighted")
print ("Precision:", precision)
print ("Recall:", recall)
```

```
Precision: 0.8658898835170666
Recall: 0.8625638075409328
```

Figure 6.1.2.7. Precision and Recall score – Decision Tree Classifier Model

```
from sklearn.neighbors import KNeighborsClassifier
k=KNeighborsClassifier()
#k.fit(x2,y2)
k.fit(x2,train_lbl2)
```

```
KNeighborsClassifier()
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(test_lbl2,result))
```

```
0.8903857217215023
```

Figure 6.1.2.8: Validating Models - K-NN Classifier Model

```
precision = precision_score(test_lbl2, result, average="weighted")
recall = recall_score(test_lbl2, result, average="weighted")
print ("Precision:", precision)
print ("Recall:", recall)
```

```
Precision: 0.8864130368093883
Recall: 0.8903857217215023
```

Figure 6.1.2.9: Precision and Recall Score - K-NN

```
class_names = test_lbl2.ProtocolName.unique()

print('classification_report =', classification_report(test_lbl2, result, target_names=class_names))
```

Figure 6.1.2.10: Classification report – K-NN

Table 6.1.2.2: Classification Report – K-NN

	precision	recall	f1-score	support
WINDOWS_UPDATE	0.97	0.94	0.95	21174
YOUTUBE	0.85	0.71	0.78	1879
HTTP_PROXY	0.64	0.43	0.52	317
HTTP	0.55	0.40	0.46	287
GOOGLE	0.00	0.00	0.00	3
SKYPE	0.00	0.00	0.00	3
AMAZON	0.00	0.00	0.00	5
FACEBOOK	0.00	0.00	0.00	13
HTTP_CONNECT	0.96	0.94	0.95	3672
SSL	0.00	0.00	0.00	4
GMAIL	1.00	0.97	0.98	2089
DROPBOX	1.00	0.07	0.12	15
TWITTER	0.91	0.92	0.92	426
MICROSOFT	0.88	0.78	0.82	6048
MSN	0.80	0.47	0.59	159

CLOUDFLARE	0.81	0.27	0.40	308
DNS	0.50	0.10	0.16	21
YAHOO	0.88	0.80	0.84	7176
WIKIPEDIA	0.40	0.33	0.36	6
WHATSAPP	0.57	0.06	0.12	62
CONTENT_FLASH	0.40	0.12	0.19	9861
APPLE_ICLOUD	0.86	0.92	0.89	234672
INSTAGRAM	0.17	0.01	0.01	180
SSL_NO_CERT	0.00	0.00	0.00	4
GOOGLE_MAPS	0.99	0.99	0.99	167334
APPLE	0.75	0.74	0.75	78035
FTP_DATA	0.90	0.27	0.42	135
IP_ICMP	0.85	0.85	0.85	152537
EASYTAXI	0.91	0.61	0.73	598
OFFICE_365	1.00	1.00	1.00	415
SPOTIFY	0.00	0.00	0.00	2
MS_ONE_DRIVE	0.00	0.00	0.00	4
APPLE_ITUNES	0.00	0.00	0.00	2
NETFLIX	0.00	0.00	0.00	3
EBAY	0.94	0.91	0.93	13369
TEAMVIEWER	0.57	0.11	0.19	71
UBUNTUONE	0.93	0.70	0.80	3541
MQTT	0.00	0.00	0.00	9
HTTP_DOWNLOAD	0.81	0.43	0.56	427
RTMP	0.77	0.27	0.40	368
TOR	0.00	0.00	0.00	1
CITRIX_ONLINE	0.94	1.00	0.97	33
DEEZER	0.85	0.73	0.78	1446
IP_OSPF	0.00	0.00	0.00	1
EDONKEY	0.00	0.00	0.00	2
NTP	0.00	0.00	0.00	2
MSSQL	0.00	0.00	0.00	1
UNENCRYPED_JABBER	0.00	0.00	0.00	4
ORACLE	0.00	0.00	0.00	1
SSH	0.85	0.63	0.72	7533
WAZE	0.00	0.00	0.00	1
TELEGRAM	0.86	0.60	0.71	306
TIMMEU	0.40	0.21	0.28	19
H323	0.95	0.97	0.96	99365
LASTFM	0.94	0.38	0.55	211
CITRIX	0.00	0.00	0.00	4
BGP	0.89	0.55	0.68	122
UPNP	0.00	0.00	0.00	6
OPENVPN	0.00	0.00	0.00	8
LOTUS_NOTES	0.53	0.11	0.19	71
TWITCH	0.00	0.00	0.00	3
FTP_CONTROL	0.77	0.50	0.61	4528
WHOIS_DAS	0.92	0.58	0.71	62
SNMP	0.00	0.00	0.00	11
TEAMSPEAK	0.00	0.00	0.00	7
MAIL_IMAPS	0.83	0.29	0.43	17
SKINNY	0.74	0.45	0.56	1135
CNN	0.00	0.00	0.00	2
BITTORRENT	0.98	0.41	0.58	552
OSCAR	0.91	0.88	0.89	8509
NFS	0.98	0.79	0.88	5265

```
   OPENSIGNAL       0.92      0.92      0.92     41610

     accuracy                           0.89    876072
    macro avg       0.51      0.36      0.40    876072
 weighted avg       0.89      0.89      0.89    876072
```

```
print('confusion_matrix =', confusion_matrix(test_lbl2, result))
```

Figure 6.1.2.11: Confusion Matrix – K-NN

Table 6.1.2.3: Confusion Matrix

```
[[19804    34     3 ...     3     0     0]
 [   57  1342    63 ...     2     0     0]
 [    3    97   136 ...     0     0     0]
                  ...
 [    0     1     0 ...  7450     1   549]
 [    0     0     0 ...     2  4174   369]
 [    2     1     0 ...   341    26 38313]]
```

```
f1 score macro 0.4039585781674261
f1 score micro 0.8903857217215023
hamming_loss 0.10961427827849765
zero_one_loss 0.10961427827849768
matthews_corrcoef 0.868500936362861
```

Figure 6.1.2.13. Results for K-NN

The classification model used after training the model and analysing the accuracies as well as speed is the Random Forest classifier. Plot tree was charted to further understand how the algorithm makes it decisions as shown in Figure 6.1.2.14.

```
from sklearn.tree import plot_tree

fig = plt.figure(figsize=(15, 10))
plot_tree(model.estimators_[0],
          feature_names=train_img2.columns,

          filled=True, impurity=True,
          rounded=True)
```

```
[Text(482.8768000094125, 538.9932203389831, 'L7Protocol <= 1.679\ngini = 0.837\nsamples = 415529\nvalue = [0, 15933, 1393, 18
9, 245, 0, 2, 3, 4, 2662, 3\n1544, 15, 330, 4615, 180, 208, 20, 5379, 7, 56\n7322, 176020, 164, 8, 125143, 58028, 100, 11415
4, 451\n280, 1, 2, 10147, 75, 2584, 2, 300, 276, 14, 1158\n3, 4, 2, 0, 1, 5757, 227, 28, 74920, 148, 0, 99\n5, 3, 50, 5, 332
9, 42, 4, 10, 17, 847, 5, 320\n6640, 4079, 31492]'),
 Text(139.85337129664245, 529.7796610169491, 'L7Protocol <= -0.839\ngini = 0.835\nsamples = 364166\nvalue = [0, 15873, 1387,
188, 245, 0, 2, 3, 4, 2662, 3\n1544, 15, 292, 4382, 180, 202, 18, 5326, 7, 54\n7003, 170585, 155, 6, 62062, 56517, 78, 10860
9, 442\n278, 1, 2, 10075, 75, 2357, 2, 295, 257, 14, 1137\n3, 4, 2, 0, 1, 5362, 227, 26, 72123, 147, 0, 99\n5, 3, 50, 5, 322
0, 42, 4, 9, 17, 837, 3, 314\n6490, 3865, 30665]'),
 Text(19.977876768269834, 520.5661016949152, 'Flow.IAT.Min <= -0.752\ngini = 0.512\nsamples = 41866\nvalue = [0, 16, 19, 7,
2, 0, 0, 0, 0, 0, 0, 219, 0\n20, 615, 0, 0, 6, 182, 7, 0, 440, 3343, 4, 2\n44952, 1364, 30, 4204, 9, 0, 0, 0, 1, 0, 677, 0\n
0, 2, 0, 0, 0, 0, 0, 0, 0, 219, 0, 8, 6920\n19, 0, 15, 0, 0, 0, 0, 189, 0, 3, 0, 0, 11\n0, 0, 7, 937, 1198]'),
 Text(10.783303821262846, 511.3525423728814, 'Total.Length.of.Bwd.Packets <= 2.388\ngini = 0.806\nsamples = 5356\nvalue = [0,
11, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5\n193, 0, 0, 0, 8, 0, 0, 165, 1366, 3, 2, 2210\n200, 0, 1492, 0, 0, 0, 0, 1, 0, 24, 0,
0, 2\n0, 0, 0, 0, 0, 0, 0, 0, 100, 0, 0, 2107, 3, 0\n15, 0, 0, 0, 0, 71, 0, 2, 0, 0, 5, 0, 0, 4\n66, 343]'),
 Text(6.546659623027179, 502.13898305084746, 'Flow.Packets.s <= -0.343\ngini = 0.813\nsamples = 4844\nvalue = [0, 11, 2, 0,
0, 0, 0, 0, 0, 0, 0, 0, 5\n186, 0, 0, 0, 8, 0, 0, 164, 1361, 3, 2, 1481\n193, 0, 1471, 0, 0, 0, 0, 1, 0, 24, 0, 0, 2\n0,
0, 0, 0, 0, 0, 0, 100, 0, 0, 2091, 3, 0\n15, 0, 0, 0, 0, 71, 0, 2, 0, 0, 5, 0, 0, 4\n66, 331]'),
 Text(2.017928637885786, 492.9254237288136, 'Flow.Packets.s <= -1.352\ngini = 0.698\nsamples = 2123\nvalue = [0, 10, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0\n156, 0, 0, 0, 3, 0, 0, 33, 231, 0, 2, 708, 15\n0, 351, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0\n0, 0,
```

Figure 6.1.2.14: Plot tree to chart the decision structure for the Random Forest classifier.

6.1.3 Step 3: Group applications or sevices – New application service classification

New applications on the network or applications on the organisation network that requires a subsliced network slice will be fed into the model and Naïve Bayes machine learning algorithm is used to classify the application to the right group. Naïve Bayes uses probability to calculate the probability of the application belonging to a certain group and predicts based on the highest probability outcome. The probability formula in Figure 6.1.3.1 is used to work out the probability of an application belonging to the same group that can share the same network resource. For example, the probably of a new online application on the network to be a video streaming application that requires high bandwidth (A) if data rate measured is high (B) will be equal to probability of high data rate (B) if high bandwidth (A) is true times the probability of high bandwidth being true (A) divided by probability of high data rate being true(B).

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Figure 6.1.3.1: Naïve Bayes Formula

Reinforcement learning algorithm is used to learn, predict, and improve the classification of application service features on the enterprise network. Entirely new applications can come online without previous data or knowledge of the environment and the proposed model will smartly classify the application feature to efficiently orchestrate the required network slice sub slice. The decision-making policy of the model is continually updated with the rewards or punishment as the agent takes actions in that state. This optimises the model's long-term reward and increases its performance and accuracy.

This improves the performance of the framework and reduces the resource required to frequently run the LazyPredict to retune the machine learning algorithm and reclassification of features.

6.1.3.1 Tuning Model

The analysed model is then further tuned to improve its accuracy. Figure 6.1.3.2 shows its parameters before tuning with associated score. Training had a perfect score of 1 before tuning and with test dataset it has a score of 0.88 before tuning the model. Figure 6.1.3.3 shows the parameter for tuning and the resulting score.

```
tunealg = ensemble.RandomForestClassifier() #Select the algorithm to be tuned
tunealg.fit(train_img2, train_lbl2)

print('BEFORE tuning Parameters: ', tunealg.get_params())
print("BEFORE tuning Training w/bin set score: {:.2f}". format(tunealg.score(train_img2, train_lbl2)))
print("BEFORE tuning Test w/bin set score: {:.2f}". format(tunealg.score(test_img2, test_lbl2)))
print('-'*10)
```

```
BEFORE tuning Parameters:  {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None,
'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_
samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': No
ne, 'verbose': 0, 'warm_start': False}
BEFORE tuning Training w/bin set score: 1.00
BEFORE tuning Test w/bin set score: 0.88
----------
```

BEFORE tuning Parameters: {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}
BEFORE tuning Training w/bin set score: 1.00
BEFORE tuning Test w/bin set score: 0.88

Figure 6.1.3.2: Parameter before tuning the Random Forest classifier model.

6.1.3.2 Subsliced Network Slice Characteristics

Based on extracted features and identified characteristics, Table 6.1.3.2.1 shows the characteristics of the various slices and type of their measured value. This forms part of the network template for initiating the network slice instance.

Table 6.1.3.2.1: Extracted network slice characteristics and measurement scale

Feature	Thresholds	Average extracted threshold	Unit of measurement
Latency	High, Medium, Low	100 milliseconds (ms) or lower. Medium to low latency needs.	milliseconds (ms) or microseconds (µs)
Bandwidth	High, Medium, Low	100 Mbps or higher Medium to High Bandwidth	kilobits per second (Kbps) or megabits per second (Mbps) or gigabits per second (Gbps) or terabits per second (Tbps)
Data volume processing (high computing-intensive systems)	High, Medium, Low	processing capacities range from gigabytes (GB) to terabytes (TB). Low to Medium	terabyte (TB) or petabyte (PB)
Spectral efficiency	High, Medium, Low	Up to 3 bps/Hz	bits per second per Hertz (bps/Hz)

Data rate	Downlink, Uplink	100 Mbps to 1 Gbps. Medium to high data rate to support diverse applications on enterprise and small business networks.	Bit/s, Mbps, Gbps
Reliability	High, Medium, Low	Reliability and availability defined by Service Level Agreements (SLAs) as uptime percentage of three nines (99.9% or higher)	dimensionless value between 0 and 1,
Throughput	High, Medium, Low	Above 1 Mbps. Medium to very high throughput	Mbps or Gbps
Speed	High, Medium, Low	100 Mbps or higher. Medium to high speed.	Mbps or Gbps
Energy efficiency	High, Medium, Low	up to 10 J/bit	Joule per bit (J/bit)
Packet loss	High, Medium, Low	Less than 5% with higher consideration for less than 1%. Medium to low	Ratio
Jitter	High, Medium, Low	Up to 50 ms. Low jitter	milliseconds (ms) or microseconds (µs).
Packet delay	High, Medium, Low	up to 100 ms	milliseconds (ms)

6.1.4 Step 4: Understand trend for classification using – Explanatory Algorithms (SHAP or LIME)

The importance of the feature and relation to the network slice characteristics was interpreted using two machine learning interpretability tools to compare and validate results. SHAP [62] and LIME [63] interpretability tools were used in this research to better understand how the model makes its decision. SHAP uses shapley value to calculate feature importance and it is a combination of multiple algorithms unlike LIME. The overall features importance in the model was determined to understand the model and key contributors. The python code in Figure 6.1.4.1 was used to plot the chat outputted in Figure 6.1.4.2.

```
feat_importances = pd.Series(clf.feature_importances_, index = test_img2.columns)

feat_importances.nlargest(12).plot(kind = 'barh')
```

Figure 6.1.4.1: Code for generating the global variable importance.

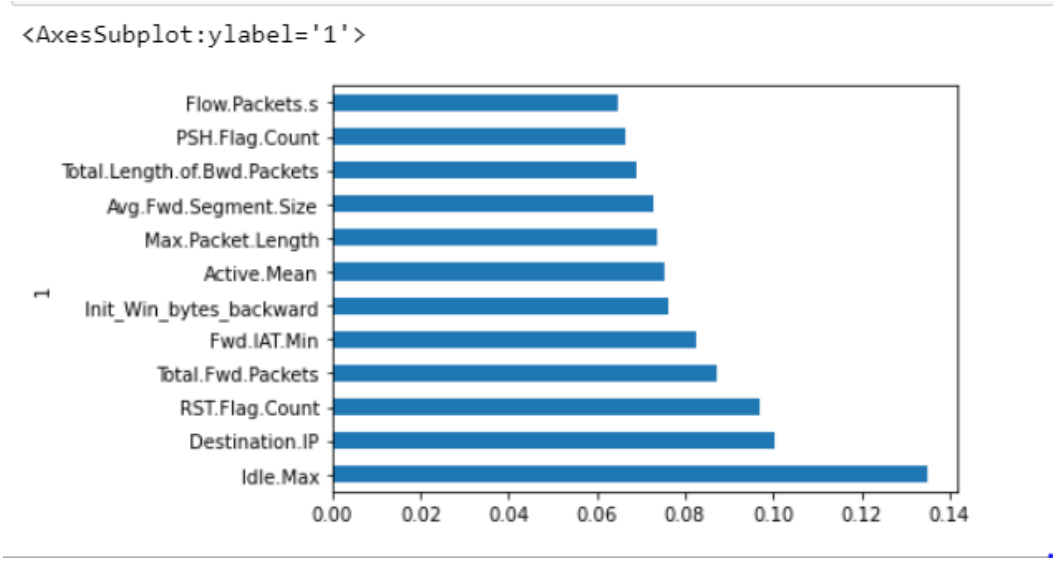


Figure 6.1.4.2: Global variable importance

Idle.Max was considered important in the model to determine active flows and which flow, or service require active network slice. Destination IP was considered another top priority feature to determine which type of service is being called and urgency, latency, as well as throughput that might be associated with that service. RST.Flag.Count was also considered highly important by the model to determine how often a TCP connection gets reset or terminated. This will enable slice to be prioritised appropriately and understand the needs of the calling application service. The total forward packets sent in a flow in the forward direction, the forward interarrival time mean (Fwd.IAT.Min) which check the minimum time between two sent packets in the forward direction. The total number of bytes sent in the backward direction in the initial window (init_Win_bytes_backward), the mean duration it took a flow to be active before becoming idle (Active.Mean) and the maximum size of a packet (Max.Packet.Length) were all classified as important features that contributed to the model's decision making in determining characteristics such as bandwidth requirement, security, latency, jitters, throughput and other network slice features for forming the network functions. Other features that contributed to the model decision's making as a matter of priority are the average packet size observed in the forward direction (Avg.Fwd.Segment.Size) and the total size of packets sent backward (Total.Length.of.Bwd.Packets). The PSH.Flag.count was also identified as part of the global importance feature to determine latency or urgency and count the number of times a flow was requested to be sent without waiting for the buffer to be full. The number of flow packets sent per second was also another identified global importance feature for the model (Flow.Packets.s). It can be deduced from these characteristics that application services on the network with large packet size and high volumes require a slice with high data rates. Flows with deep coverage will require an ultra-high-density slice and will be devices with ultra-low energy consumption requirements. Flows that are highly active with high packet transfer per second with very low data loss will require ultra-low latency slice with high reliability and mobility. These features help in formulating the network slice template and identifying groups of application services on the smart enterprise network that will share the subsliced network functions. Table 6.1.4 summarises the requirements of the typical types of applications on the company network and associated typical type of slices based on the findings from the research on types of slices allocated to ProtocolName. This is used in the eSS model as the fall-back default slice to orchestrate for the specified application if the network resource is not enough to meet the original

request. If this can't be satisfied, the service function then looks for another alternative network functions to be chained with lesser network resource requirements. If this can't be fulfilled, then autoscaling or queuing is considered.

1Table 6.1.4 Application Types with Fall Back Subslice Feature

Type of Application Available on Company Network	Characteristics of the Subslice
Email, Web Browsing and General Internet Use	Medium Packet Delay Low Speed (Typically less than 1 Mbps) Low Data Volume Processing (Typically less than 1 terabyte (TB)) Medium Latency (50-100 ms) Low to Medium Throughput (Up to 100 Mbps) Low Jitter (Up to 50ms)
VoIP (Voice over Internet Protocol)	Low Packet Delay (Typically less than 50 milliseconds (ms)) Low Packet Loss (Typically less than 1%) Low Latency (0-50 ms) Medium Throughput (1 Mbps to 100 Mbps) Low Jitter (Up to 30ms)
video conferencing	Low Packet Delay (Typically less than 50 milliseconds (ms)) Low Packet Loss (Typically less than 1%) Low Latency (0-50 ms) Medium Throughput (1 Mbps to 100 Mbps) Low Jitter (Up to 30ms)
online gaming.	Low Packet Delay (Typically less than 50 milliseconds (ms)) High Packet Delay (Typically between 100 ms and 500 ms) High Speed (Typically between 100 Mbps and 1 Gbps (1,000 Mbps)) Low Latency (0-50 ms) Medium Throughput (1 Mbps to 100 Mbps) High Data Rate Low Jitter (Up to 30ms)
Video Streaming	Medium Packet Loss (Typically between 1% and 5%) Low to Medium Packet Delay (Up to100 ms) High Speed (Typically between 100 Mbps and 1 Gbps (1,000 Mbps)) Medium to High Throughput (1 Mbps to 100 Mbps) High Data Rate Low to Medium Jitter (up to 100 ms)
Large File Downloads.	High Speed (Typically between 100 Mbps and 1 Gbps (1,000 Mbps)) High Throughput (Above 1 Gbps) Low to Medium Packet Delay (up to 100 ms) Low to Medium Jitter (up to 100 ms)
Cloud Services and Remote Desktop	Low to Medium Packet Delay (up to 100 ms) Low to Medium Jitter (up to 100 ms) Low Latency (0-50 ms) Flexible Data Rate

IoT Sensors and Industrial Automation	Low Data Rate (Up to 1 Mbps) Low Power Consumption
High-Volume Data Streaming	Low Latency (0-50 ms) High Throughput (Above 1 Gbps)
Autonomous Vehicles	Ultra-Low Latency (single-digit millisecond range) High Reliability
Bulk Data Transfer (Backup)	High Data Rate (Above 1 Gbps) Moderate Latency
Video Surveillance	High Data Rate (Above 1 Gbps) Low Latency (0-50 ms) Low Packet Loss (Less than 1%)
Critical Infrastructure Control	Low Latency (0-50 ms) High Reliability

	Packet Delay		Speed		Data Volume	Latency			Throughput			Jitter		Packet Loss			
Type of Application Available on Company Network	Low Packet Delay (Typically less than 50 milliseconds (ms))	Medium Packet Delay	Low Speed (Typically less than 1 Mbps)	High Speed (Typically between 100 Mbps and 1 Gbps (1,000 Mbps))	Low Data Volume Processing (Typically less than 1 terabyte (TB))	Ultra-Low Latency(single-digit millisecond range)	Low Latency (0-50 ms)	Medium Latency (50-100 ms)	Low to Medium Throughput (Up to 100 Mbps)	Low to Medium Throughput (Up to 100 Mbps)	High Throughput (Above 1 Gbps)	Low Jitter (Up to 50ms)	Medium Jitter (up to 100 ms)	Low Packet Loss (Typically less than 1%)	Medium Packet Loss (Typically between 1% and 5%)	Low Power Consumption	High Reliability
Email, Web Browsing and General Interne	x	Y	Y	x	Y	x	x	Y	Y	Y	x	Y	x	Y	x	x	x
VoIP (Voice over Internet Protocol)	Y	x	x	x	x	x	Y	x	x	Y	x	Y	x	x	x	x	x
Video Conferencing	Y	x	x	x	x	x	Y	x	x	Y	x	Y	x	Y	x	x	x
Online Gaming.	Y	Y	x	Y	x	x	Y	x	x	Y	x	Y	x	x	x	x	x
Video Streaming	Y	Y	x	Y	x	x	x	x	x	Y	Y	Y	Y	x	Y	x	x
Large File Downloads	Y	Y	x	Y	x	x	x	x	x	x	Y	Y	Y	x	x	x	x
Cloud Services and Remote Desktop	Y	Y	x	x	x	x	Y	x	Y	Y	Y	Y	Y	x	x	x	x
IoT Sensors and Industrial Automation	x	x	x	x	x	x	x	x	Y	x	x	x	x	x	x	Y	x
High-Volume Data Streaming	x	x	x	x	x	x	Y	x	x	x	Y	x	x	x	x	x	x
Autonomous Vehicles	x	x	x	x	x	Y	x	x	x	x	x	x	x	x	x	Y	x
Bulk Data Transfer (Backup)	x	x	x	x	x	x	x	Y	x	x	Y	x	x	x	x	x	x
Video Surveillance	x	x	x	x	x	x	Y	x	x	x	Y	x	x	Y	x	x	x
Critical Infrastructure Control	x	x	x	x	x	x	Y	x	x	x	x	x	x	x	x	x	Y

Figure 6.1.4.3 Application Type Mappings with Default Network Characteristics Mappings

Table 6.1.4 and Figure 6.1.4.3 show the summary of the default selected subslice used in the proposed algorithm in this study. This proposed “Resource Alternative Selection Algorithm” aims to improve the performance of selecting subslices with lesser resource demands and efficiently allocating resources to reduce chances of delay or denial. Some of the application types that were monitored on the given dataset in this study and how they were classified per metric measured is shown in Figure 6.1.4.3. It shows the mapping of application types to QoS metric thresholds (as detailed in Section 8.3), showing how different applications are supported under varying network conditions.

The eSS analyses the real time network traffic, classifies, and identifies the required subslices as well as the required network resources to orchestrate the slice. The framework continuously checks the available resource against requested network resources and intelligently allocates the network resource to meet the needs of all active device on the network if possible. If the framework returns an output indicating the available resource might not be sufficient, the proposed “Resource Alternative Selection Algorithm” is triggered to efficiently select an alternative defined network characteristics with lesser resource utilisation to fulfil the network requirement of the application. If the available network resource is still not enough, then the algorithm tries to use the default characteristics Json to select the minimum allowed resource for that characteristic on the company network. If this still fails, then the algorithm returns “No Suitable Slice Found”. This output can then trigger another event within the framework to auto scale instead or followed a defined path as programmed. The algorithm maps the application characteristics to preferred network slices, and the most suitable slice based on the required characteristics and application type is then selected. Figure 6.1.4.4 shows the proposed Resource Alternative Selection Algorithm written in the python programming language however any language can be used.

```python
def select_network_slice(application_type, characteristics, default_characteristics):
    # Defining the available network slices and their characteristics
    network_subslices = {
        "Email and Web Browsing": {
            "Latency": "Medium",
            "Throughput": "Balanced"
        },
        "IoT Sensors": {
            "Power Consumption": "Low",
            "Throughput": "Low"
        },
        "Voice over IP (VoIP)": {
            "Latency": "Low",
            "Packet Loss": "Low"
        },
        "Real-Time Gaming": {
            "Reliability": "High",
            "Latency": "Low",
            "Jitter": "Low"
        },
        "High-Security Data Transfer": {
            "Security": "High",
            "Latency": "Low",
            "Packet Loss": "Low"
        },
        "Video Conferencing": {
            "Latency": "Low",
            "Security": "Medium",
            "Throughput": "High"
        },
         "Video Surveillance": {
            "Latency": "Low",
            "Packet Loss": "Low",
            "Throughput": "High"
        },
        "Rural Area Coverage": {
            "Deep Coverage": True,
            "Throughput": "Balanced"
        },
        "Telemedicine": {
            "Latency": "Low",
            "Packet Loss": "Low",
            "Reliability": "High"
        },
        "Critical Infrastructure Control": {
            "Latency": "Low",
            "Security": "High",
            "Reliability": "High"
        },
        "Energy-Efficient IoT": {
            "Power Consumption": "Low",
            "Throughput": "Low"
        },
```

```
    },
    "Autonomous Vehicles": {
        "Latency": "Ultra-Low",
        "Reliability": "High"
    },
    "Bulk Data Transfer (Backup)": {
        "Latency": "Medium",
        "Throughput": "High"
    },
    "High-Bandwidth Data Transfer": {
        "Bandwidth": "High",
        "Latency": "Low"
    },
    "Public Wi-Fi Hotspots": {
        "Throughput": "Balanced",
        "Security": "Medium"
    },
    "Spectrally Efficient Services": {
        "Spectral Efficiency": "High",
        "Latency": "Medium"
    },
    "Cloud-Based Applications": {
        "Throughput": "Medium",
        "Latency": "Low"
    },
    # More network slice characteristics can be added as needed by the company network

}

# Loading predefined default characteristics with likely lesser network resource requirement from JSON
default_characteristics = json.loads(default_characteristics_json)

# Defining the mapping of characteristics to slice preferences
characteristics_mapping = {
    "Deep Coverage": ["Rural Area Coverage"],
    "Security": ["High-Security Data Transfer"],
    "Jitter": ["Real-Time Gaming"],
    "Latency": ["High-Security Data Transfer", "Real-Time Gaming"],
    "Packet Loss": ["High-Security Data Transfer"],
    "Reliability": ["Real-Time Gaming"],
    "Data Rate": ["High-Security Data Transfer", "Rural Area Coverage"],
    # More characteristic preferences can be added as needed by the company network
}

# Initializing a list of candidate slices based on the characteristic preferences
candidate_subslices = []

for characteristic, preference in characteristics.items():
    if preference == "None":
        continue  # Skiping any characteristics with no preference

    if characteristic in characteristics_mapping:
        candidate_subslices.extend(characteristics_mapping[characteristic])
```

```
    # Filtering the candidate slices based on the active application type
    suitable_subslices = [
        subslice_name for subslice_name, slice_characteristics in candidate_subslices.items()
        if subslice_name == application_type or subslice_name == "General Purpose"  # Including a default slice for unmatched applications
    ]

    # Prioritizing slices based on the number of matching characteristics
    prioritised_subslices = sorted(
        suitable_subslices,
        key=lambda subslice_name: sum([characteristics[s] == network_subslices[subslice_name][s] for s in characteristics]),
        reverse=True
    )

    # Checking if there are network resources to fulfill the selected subslice's requirements
    selected_subslice = None

    for subslice_name in prioritized_slices:
        sub_slice_requirements = network_subslices[subslice_name]
        can_fulfill = all([characteristics[characteristic] == sub_slice_requirements[characteristic] for characteristic in characteristics])

        if can_fulfill:
            selected_subslice = subslice_name
            break


    # Pprioritising default characteristics If no suitable subslice is found
    if selected_subslice is None and default_characteristics:
        for characteristic in default_characteristics:
            if characteristics[characteristic] == "None":
                characteristics[characteristic] = default_characteristics[characteristic]

        # Sorting default characteristics by network resource requirements (lower values are prioritized)
        default_characteristics = sorted(default_characteristics, key=lambda k: network_subslices[application_type][k])

        # Trying to select a suitable subslice with default characteristics
        for characteristic in default_characteristics:
            selected_subslice = select_network_slice(application_type, characteristics, {})

            if selected_subslice != "No Suitable Slice Found":
                break

    if selected_subslice is None:
        return "No Suitable Slice Found"
    else:
        return selected_subslice
```

```
# How to  use:
application_type = "Telemedicine"
characteristics = {
    "Latency": "Low",
    "Packet Loss": "Low",
    "Reliability": "High""
}

default_characteristics_json = '''
{
    "Reliability": "Medium",
    "Jitter": "Low",
    "Latency": "Medium",
    "Throughput": "Medium",

}
'''


selected_subslice = select_network_slice(application_type, characteristics, default_characteristics)
print("Selected Network Slice:", selected_subslice)
```

Figure 6.1.4.4 Resource Alternative Selection Algorithm

6.1.4.1. SHAP

Shapeley calculated the contribution of each feature in the model prediction using a cooperative gaming concept [64]. SHapley Additive exPlanations (SHAP) is built on Shapely values. Shapely based the cooperative game on four main accepted truth:

- Accepted Truth 1: Efficiency. Shapely gives all features or players in his gaming concept a fair chance. Each feature contributes to the final coalition. The sum of each feature shapely value makes up the total coalition value.
- Accepted Truth 2: Symmetry. It is a fair game with equal chance for each feature or player in the game.

- Accepted Truth 3: Dummy. In the event whereby the contribution of a feature or player i is nothing in the game to any coalition s, then the feature i contribution which is zero can be expressed as $\varphi_i(x)=o$
- Accepted Truth 4: Additivity. The arithmetic summation for a pair of game x, y can be expressed as $\varphi(x+y)=\varphi(x)+\varphi(y)$, where $(x+y)(S)=x(S)+y(S)$ for total coalition S

The contribution of each feature i can be calculated as:

$$\varphi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(N-|S|-1)!}{N!}(v(S \cup \{i\}) - v(S))$$

Where N is the total number of features or players, and S is subset of N. The total value of S features or players is $v(S)$. Feature i marginal contribution can then be expressed as $v(S \cup \{i\}) - v(S)$ when it joins S players.

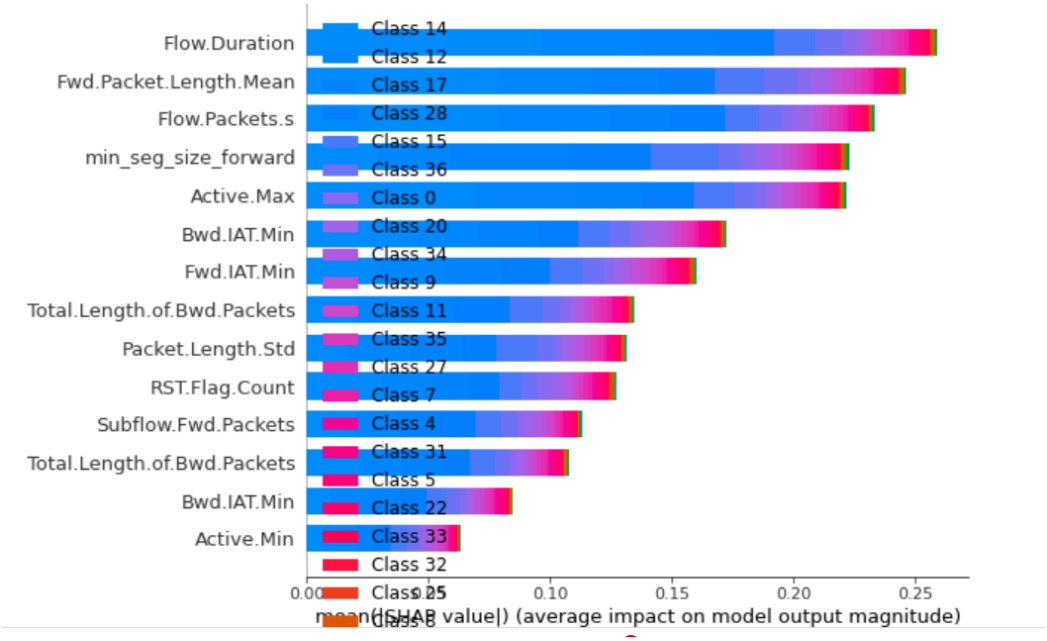
The code shown in Figure 6.1.4.1 plots the variable importance chart. This plot illustrates the importance of each feature in descending order which the model used in making its decision to predict network slice characteristics. The top one is the most significant feature with highest predictive power.

```
import shap
shap_values = shap.TreeExplainer(clf).shap_values(train_img2)
shap.summary_plot(shap_values, train_img2, plot_type="bar")
```

Figure 6.1.4.1.1: SHAP code for variable importance plot

The output of the code showing the significance of each feature is illustrated in Figure 6.1.4.2.

```
shap.summary_plot(shap_values, train_img2, plot_type="bar")
```



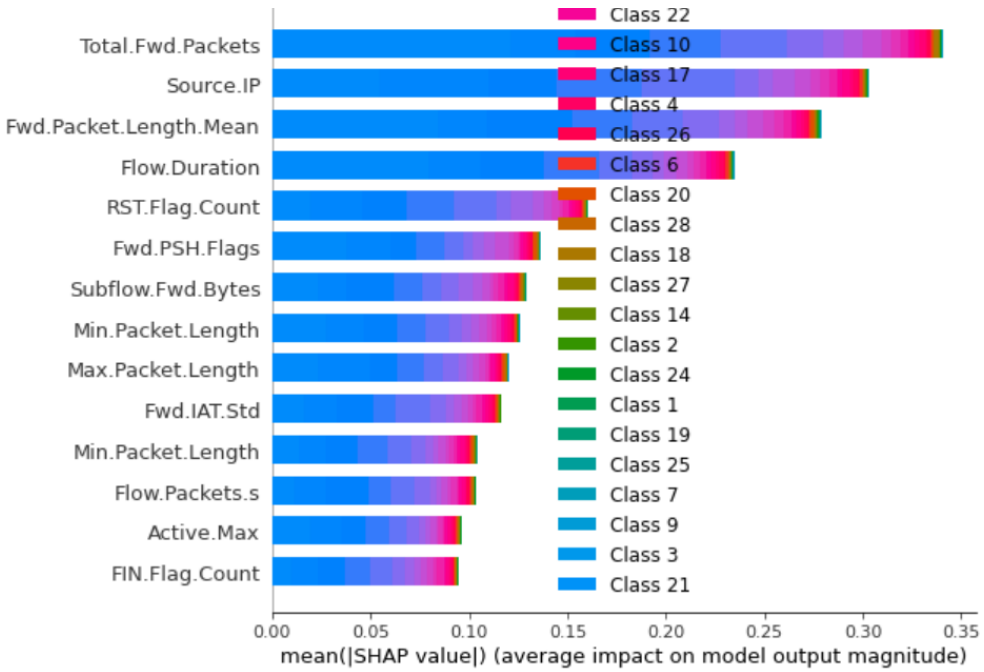


Figure 6.1.4.1.2: Variable importance plot

The positive and negative relationship are then plotted against the target application service variables. The output is shown in Figure 6.1.4.3.

```
#correlation - shap variable importance plot
shap.summary_plot(shap_values, X_train)
```

```
shap.dependence_plot("Flow.Packets.s", shap_values[0], train_img2)
```

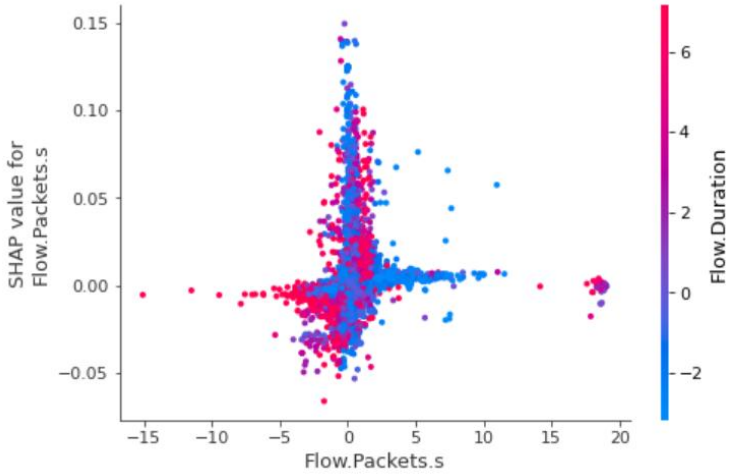


Figure 6.1.4.1.3: Correlation – SHAP variable importance plot

Combination of the vertical heat line and the x-axis determines the impact of each feature on the target variable. The x-axis plots the prediction level of the feature in relation to the target, it shows if

it is high or low prediction value. A simplified chart is shown in Figure 6.1.4.4 illustrating a positive and high correlating feature with the target variable in red.

```
##simplified version
df = train_img
def ABS_SHAP(df_shap,df):
    #import matplotlib as plt
    # Make a copy of the input data
    shap_v = pd.DataFrame(df_shap)
    feature_list = df.columns
    shap_v.columns = feature_list
    df_v = df.copy().reset_index().drop('index',axis=1)

    # Determine the correlation in order to plot with different colors
    corr_list = list()
    for i in feature_list:
        b = np.corrcoef(shap_v[i],df_v[i])[1][0]
        corr_list.append(b)
    corr_df = pd.concat([pd.Series(feature_list),pd.Series(corr_list)],axis=1).fillna(0)
    # Make a data frame. Column 1 is the feature, and Column 2 is the correlation coefficient
    corr_df.columns  = ['Variable','Corr']
    corr_df['Sign'] = np.where(corr_df['Corr']>0,'red','blue')

    # Plot it
    shap_abs = np.abs(shap_v)
    k=pd.DataFrame(shap_abs.mean()).reset_index()
    k.columns = ['Variable','SHAP_abs']
    k2 = k.merge(corr_df,left_on = 'Variable',right_on='Variable',how='inner')
    k2 = k2.sort_values(by='SHAP_abs',ascending = True)
    colorlist = k2['Sign']
    ax = k2.plot.barh(x='Variable',y='SHAP_abs',color = colorlist, figsize=(5,6),legend=False)
    ax.set_xlabel("SHAP Value (Red = Positive Impact)")

ABS_SHAP(shap_values,train_img2)
```

Figure 6.1.4.1.4: Correlation – Simplified version

The marginal effect the identified most important feature is then determined as shown in Figure 6.1.4.1.5.

```
shap.dependence_plot("Total.Length.of.Bwd.Packets", shap_values[0], train_img2)
```

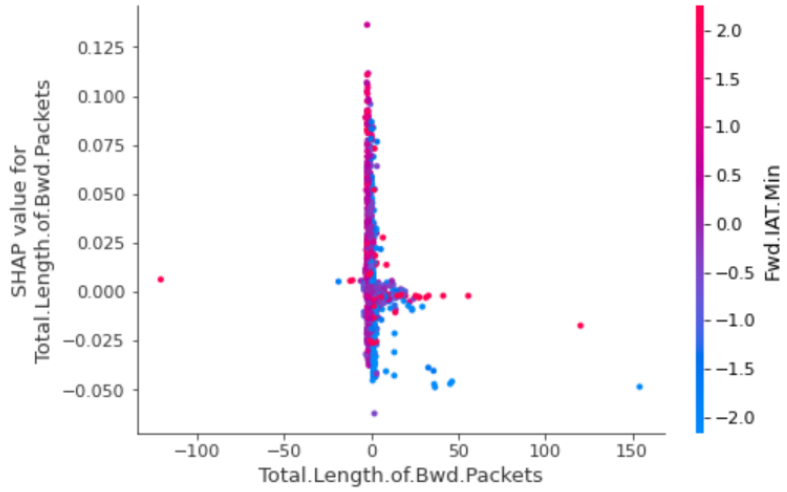


Figure 6.1.4.1.5: Dependence plot

6.1.4.2 LIME

The novel solution proposed by Ribeiro, Marco Tulio et al. [63] to validate and trust a model prediction as well as the model itself called Local table Interpretable Model-Agnostic Explanations (LIME) is used to interpret the proposed model in this research and determine trust. The interpretable model is learnt locally with LIME unlike SHAP which is done globally. LIME is a subset of SHAP [65] giving it an advantage of speed over SHAP. While Shapely values permutates all possible outcomes to achieve a global and local model interpretability and consistency, LIME plots sparse linear models to achieve a local model interpretability and consistency in its local vicinity. It builds its prediction around a single feature prediction locally. Figure 6.1.4.2.1 shows the code and output of the LIME interpretability model on the network packet dataset after applying a Random Forest regressor on the model. The chart explains the model and feature importance for each type of application service to determine the network slice characteristics. The first two application services are shown below in Figure 6.1.4.2.1 and Figure 6.1.4.2.6. The two application services shown which are the top two from the test data are Windows Update and YouTube. This is shown by the code and output from Figure 6.1.4.2.4. The predictions for Windows update and YouTube services are -0.37 and -0.16 respectively. Twelve features were used based on the selected features from the Step 1 process of building the framework model.

```
import lime
import lime.lime_tabular

explainer = lime.lime_tabular.LimeTabularExplainer(np.array(train_img2),
                    feature_names=train_img2.columns,
                    class_names=['ProtocolName'],
                    verbose=True, mode='regression')
```

```
exp = explainer.explain_instance(test_img2.iloc[0],
     model.predict, num_features=12)
exp.as_pyplot_figure()
```

```
C:\Users\abimbola.efunogbon\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names,
but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

```
Intercept -0.042508826484052464
Prediction_local [-0.4781787]
Right: -0.3705263351798679
```

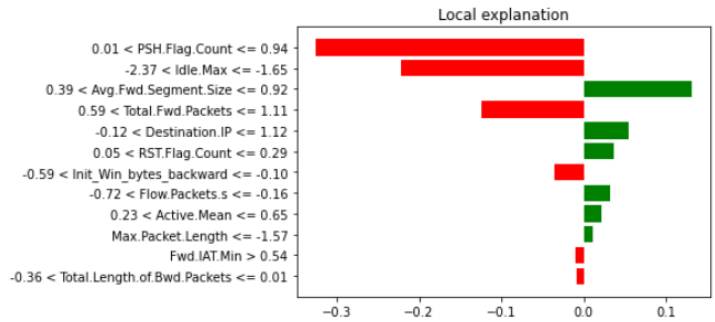


Figure 6.1.4.2.1: LIME to explain the model and predictions for Windows Update service.

The feature that has a high positive impact on the application services which is the protocol name in the dataset is shown in green while features with high negative correlation with the target are shown in red. From Figure 6.1.4.2.1, Average forward segment size greater than 0.39 and less than or equal to 0.92 is one of the characteristics to subslice the network slice for windows update service. Destination IP greater than -0.12 and less than or equal to 1.12 has a high impact on the application service. The real IP can be determined by inversing the transformed LabelEncoder value in step 1 of

the framework model design phase. The output also shows RST flag count greater than 0.05 and less than 0.29 has high positive impact on the windows update service. The PSH flag count greater than 0.01 and less than 0.94 has high negative impact on the windows update same as idle max greater than -2.37 and less than or equal to -1.65 negatively correlates with Window update service network requirements. The impact and importance are shown in descending order. The results can also be represented in notebook format for clarity as shown in Figure 6.1.4.2.2

```
exp = explainer.explain_instance(test_img2.iloc[0], model.predict)
exp.show_in_notebook(show_table=True, show_all=False)
```

```
Intercept -0.01660728072503334
Prediction_local [-0.55013184]
Right: -0.3705263351798679
```

```
C:\Users\abimbola.efunogbon\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names,
but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

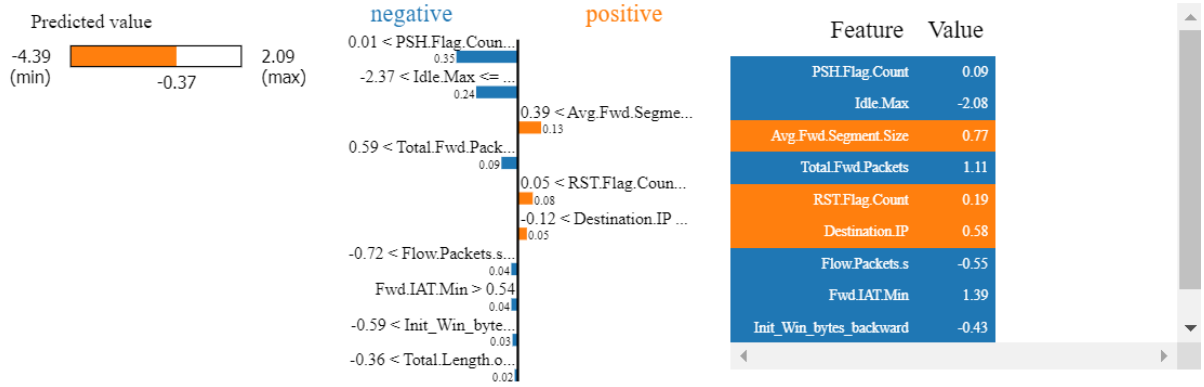


Figure 6.1.4.2.2: LIME Notebook report for Windows Update service interpretability

The LIME model intercept is -0.0166. The prediction of the LIME model is -0.55 and the original Random Forest regressor prediction is -0.37. LIME's prediction_local = sum of coefficient + intercept. The total of the coefficient can be calculated from the code shown in Figure 6.1.4.2.3 which is -0.5335.

```
pd.DataFrame(exp.as_list())[0].sum()
```

```
'0.01 < PSH.Flag.Count <= 0.94-2.37 < Idle.Max <= -1.650.39 < Avg.Fwd.Segment.Size <= 0.92-0.12 < Destination.IP <= 1.120.59 <
Total.Fwd.Packets <= 1.110.05 < RST.Flag.Count <= 0.29-0.59 < Init_Win_bytes_backward <= -0.10-0.72 < Flow.Packets.s <= -0.160.
23 < Active.Mean <= 0.65Fwd.IAT.Min > 0.54Max.Packet.Length <= -1.57-0.36 < Total.Length.of.Bwd.Packets <= 0.01'
```

Figure 6.1.4.2.3: Total Coefficient

LIME prediction = -0.0166 (Intercept) + -0.5335 (Total of coefficient) = -0.55

```
clf.predict(test_img2[0:2])
```

```
array(['WINDOWS_UPDATE', 'GOOGLE'], dtype=object)
```

```
test_lbl2[0:2]
```

```
3354790    WINDOWS_UPDATE
1064925           YOUTUBE
Name: ProtocolName, dtype: object
```

Figure 6.1.4.2.4: First two application service records predicted of the test dataset.

The list of coefficients for the LIME model is shown below in Figure 6.1.2.5.

```
pd.DataFrame(exp.as_list())
```

	0	1
0	0.01 < PSH.Flag.Count <= 0.94	-0.316522
1	-2.37 < Idle.Max <= -1.65	-0.249554
2	0.39 < Avg.Fwd.Segment.Size <= 0.92	0.125382
3	-0.12 < Destination.IP <= 1.12	0.112455
4	0.59 < Total.Fwd.Packets <= 1.11	-0.110330
5	0.05 < RST.Flag.Count <= 0.29	0.062641
6	-0.59 < Init_Win_bytes_backward <= -0.10	-0.030991
7	-0.72 < Flow.Packets.s <= -0.16	-0.018705
8	0.23 < Active.Mean <= 0.65	-0.016995
9	Fwd.IAT.Min > 0.54	0.012305
10	Max.Packet.Length <= -1.57	-0.011901
11	-0.36 < Total.Length.of.Bwd.Packets <= 0.01	-0.005213

Figure 6.1.4.2.5: List of LIME coefficients

The second data from the test dataset is YouTube application service and the impact of each feature for the model prediction of the characteristic of the slice is shown in Figure 6.1.4.2.6 below.

```
exp = explainer.explain_instance(test_img2.iloc[1],
     model.predict, num_features=12)
exp.as_pyplot_figure()
```

```
C:\Users\abimbola.efunogbon\Anaconda3\lib\site-packages\sklearn\base.py:450: U
but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

```
Intercept -0.18960654193049875
Prediction_local [0.05371218]
Right: -0.16069106455362664
```

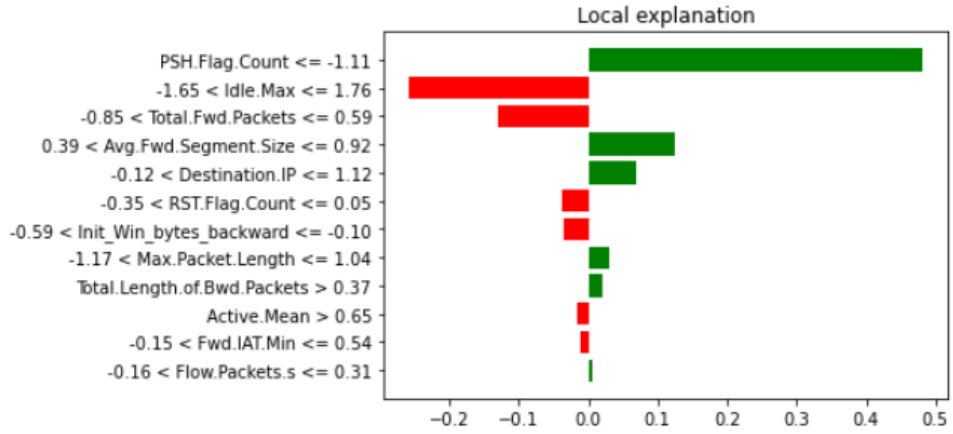


Figure 6.1.4.2.6: LIME to explain the model and predictions for YouTube service.

```
exp = explainer.explain_instance(test_img2.iloc[1], model.predict)
exp.show_in_notebook(show_table=True, show_all=False)
```

```
Intercept -0.19781218891437835
Prediction_local [0.04920022]
Right: -0.16069106455362664
```

```
C:\Users\abimbola.efunogbon\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names,
but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

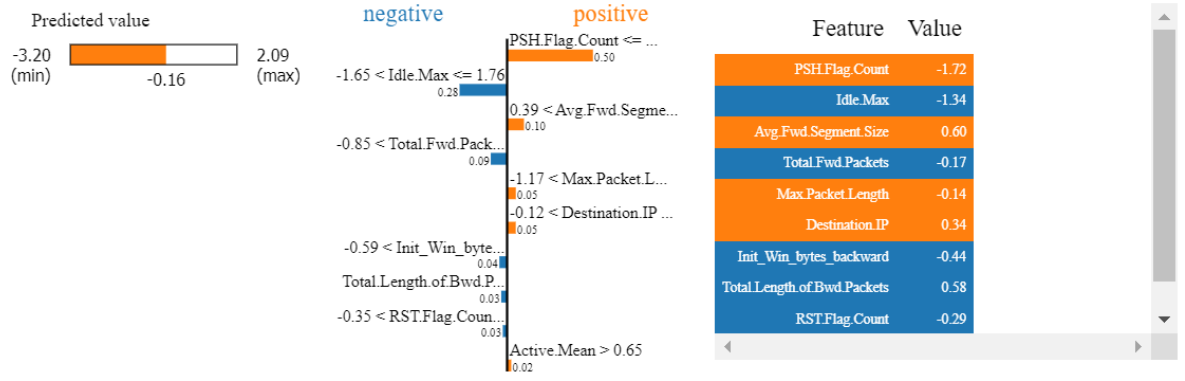


Figure 6.1.4.2.7: LIME Notebook report for YouTube service interpretability

6.1.5 Step 5: Define each slice and allocate to grouped applications

Each subsliced slice with specific network functions are grouped based on combination of the unique characteristic requirements of user equipment on the network. These then form the shared subsliced

network efficiently orchestrated to meet specific needs saving resources and cost compared to the generic network slice of uRLLC, eMBB or mMTC.

6.1.6 Step 6: Auto assign network slice to new setvices with similar features/characteristics using - Similarity Algorithms (KNN, SVD, Cosine, Euclidean Distance, Levenshtein, Jaro-Winkler, and other classification machine learning models)

The self-organised, self-optimised and efficiently orchestrated network slice is then auto assigned to the requesting user application as the eSS model output, monitored and managed in real-time.

Network slices are subsliced to efficiently share the resources based on KPIs and network requirements with the features provided by network function virtualisation and service chaining functions.

This smart company will be requesting only required network slices. The network slice subscribed to will be subsliced into the application, platform, and infrastructure-based network slices. The request will go to the 5G new radio base station which then uses the radio access network, transport network, core network and the network slice management service to efficiently slice the core layer, RAN layer and transport layer end to end into the eMBB, mIoT, uRLLC network slice with supervised machine learning algorithm. These then get sliced into the infrastructure based, Application base, and platform-based network. The management service will then further subslice the subsliced network and allocate to services requested using a classification machine learning algorithm. The slice layers can be seen in Figure 6.1.6.1. This is shown by demonstrating two sample departments.

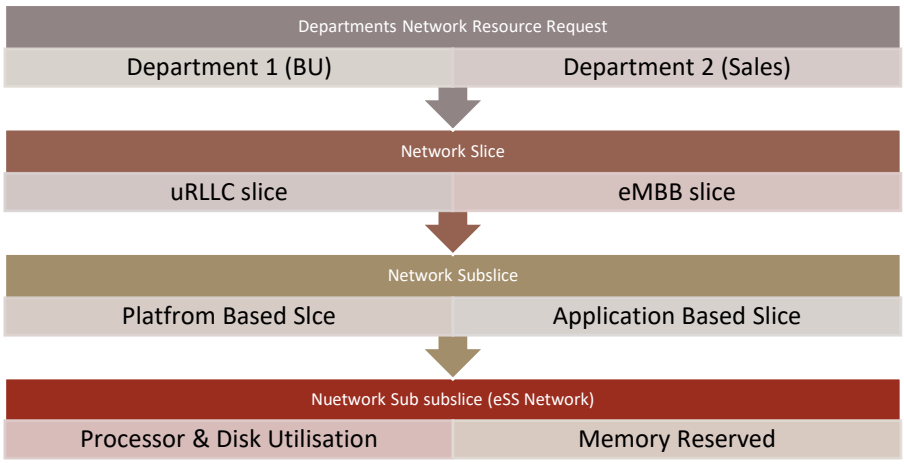


Figure 6.1.6.1: Network Slice Layer – Enhanced Subsliced Network

6.2 Proactive and Adaptive Network Slice Prediction

The traffic prediction module contributes to the efficient utilisation of network resources and better service quality in the eSS model. The module synthesizes time series forecasting technique, reinforcement learning (RL), AutoRL and RayTune to enable proactive and adaptive network slice prediction and management. The model is flexible and AutoRL can be excluded where compute resource is limited yet still achieving a scalable solution. The result comparing AutoRL with RayTune is further discussed in Chapter 8. Future traffic patterns for each network slice are predicted by the

Traffic Prediction Module by utilising specialised time-series forecasting techniques. Ray Tune is used to automatically select the most suitable RL technique to fit the analysed 5G network environment. The criteria for selecting the most suitable RL technique is the one that maximises the cumulative reward while minimising training time and resource consumption. Ray Tune evaluates each RL algorithm using these criteria through extensive cross-validation and benchmarking against the 5G environment. AutoRL can automate and optimises the development of the RL strategies to ensure the best configurations are used. The network slices are dynamically adjusted with RL in real-time based on feedback from the enterprise network. Figure 6.2.1 illustrates the integration of these tools within the traffic prediction module.

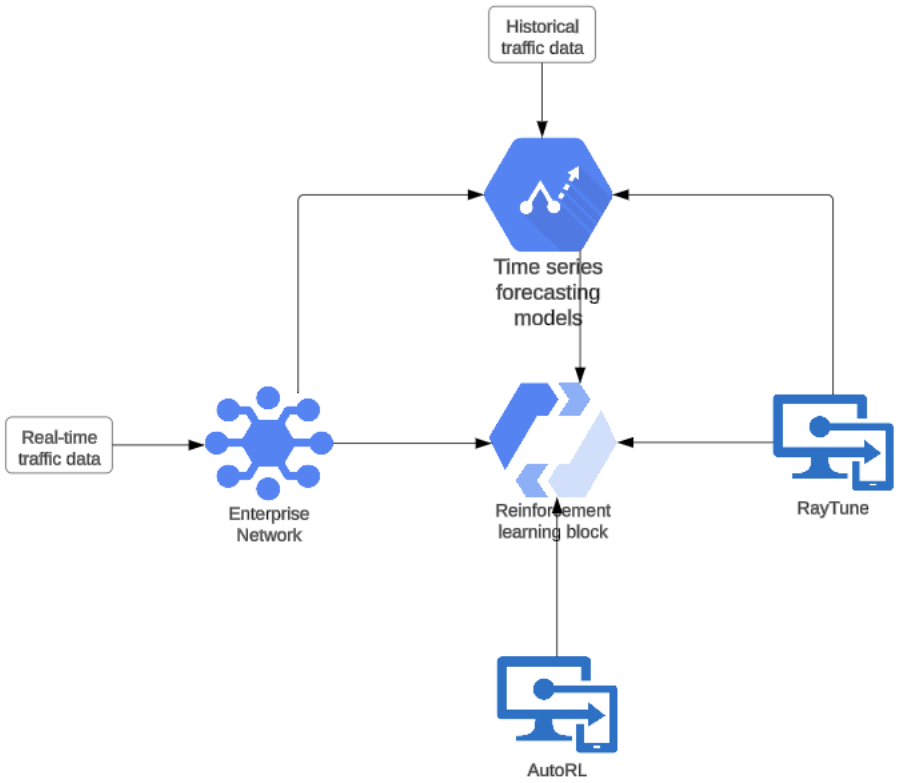


Figure 6.2.1 Traffic Prediction Module

Historical network traffic data and real-time network data from the enterprise network are fed into the time series forecasting models. The time series forecasting models then predicts future network demand for different traffic types. It then sends the forecast to a reinforcement learning block. The RL block also receives real-time network performance data from the enterprise network and dynamically adjust its strategies. The block uses the predictions from the time series model to proactively allocate resources to different slices. The RL strategies and the model's learning rates gets automatically tuned by AutoRL for automated performance optimisation. RayTune tunes both the time series forecasting model and RL model hyperparameters. It optimises the parameter for both demand prediction and resource allocation to improve accuracy and efficiency. Based on the environment network dataset used in this study, Proximal Policy Optimization (PPO) was identified as the most suitable RL technique for optimising the 5G environment and was used as the default algorithm for RayTune in the eSS model. The results are further discussed in Chapter 8 under evaluation and results.

One of the fundamental technologies applied in this research towards dynamic orchestration of the network functions on-demand is Service Function Chaining and leveraging on NFV as well as SDN.

6.3 The Service Function Chaining Enabled Domain Layer

Service Function Chaining (SFC) is a concept which chains the various connected service functions within a network system to complete a process to provide value added service to users [29]. It defines the path for the service functions. The main aim of SFC is to deploy complex and flexible service functions [29]. This SFC architecture provides flexible deployment of customised services and allows dynamic inventory of services such that service functions can be added or removed dynamically within the chain without breaking the chain. This is possible with software defined networking. The SFC will steer the network traffic through the list as defined in the architecture in Figure 6.1. The components are placed along the service function path (SFP), This architecture makes it possible to scale or swap the network topology or even isolate the service functions from the overall topology without affecting the network services [29]. The service function forwarder directs each service function traffic to the next service function in the chain through the overlay as defined in the service functions policy. Service functions proxies are placed between the service function forwarder and the service functions which are not SFC aware themselves. It helps integrate the service functions into the service function chaining by consuming and adding metadata on behalf of the service functions [29]. All the service functions are virtualised and deployed with NFV and SDN. Although service function chaining technology has been implemented in this study due to the flexibility and scalability it offers among other features, the network slice management function can be deployed into any virtualised server in any domain within the network using a different network topology.

The network slice manager deployed on the edge will be deployed in a Service Function Chaining (SFC)enabled domain. The slice manager receives and analyses the network service request. Real time data collected is compared with the predicted network slice for that request type. If a cache is deployed in this layer, it checks in the in-memory cache to reduce latency. The generic slice template received is converted to a network slice template. The service analyses this network resource request, implements the admission control mechanism, and issues network resource assignment, monitoring of the network resource utilisation which can be deployed as a microservice application within the layer. An architecture diagram of the edge layer which hosts the network slice management service is shown in Figure 6.3.1.

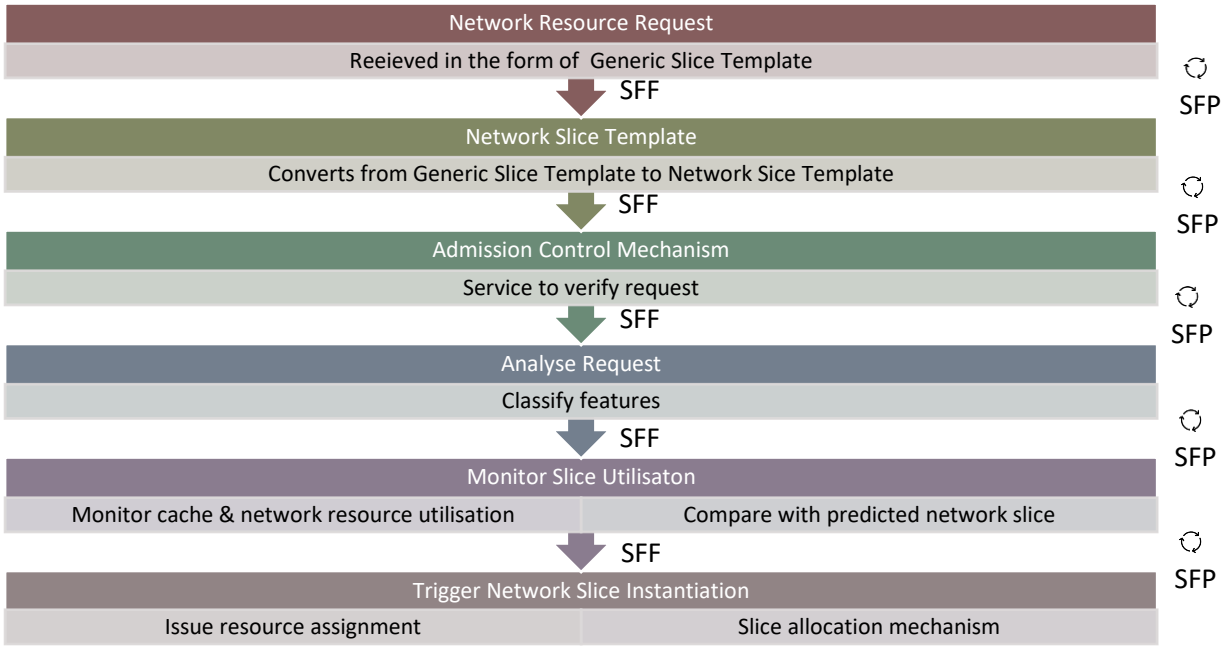


Figure 6.3.1: Network Manager Architecture

SFF = Service Function Forwarder

SFP = Service Function Path

The smart classification of the features for the enhanced subsliced network is implemented in the analyse request process. This will be a service function node within the service function chaining domain. Both the Analyse Request service function node and the Monitor Slice Utilisation service function node will interact with the machine learning model towards efficiently orchestrating the network slices. The iteration process within the model is shown in Figure 6.3.2 below.

```
Network Slice Template (NST):

  Slice ID: 123456789
  Slice Name: eMBB Slice Instance 1
  Description: Specific instance of an eMBB network slice.

  Characteristics:
    - Slice Type: Broadband Internet
    - Use Case: High-speed data services for mobile users
    - Latency Requirements: 10 ms maximum latency for real-time applications
    - Throughput: Minimum guaranteed throughput of 100 Mbps, maximum achievable throughput of 1 Gbps
    - Resource Isolation: Isolated virtual network resources
    - Security: Enhanced security measures

  Architecture:
    - Network Functions:
      - Radio Access Network (RAN)
      - Core Network (CN)
      - User Plane Function (UPF)
      - Session Management Function (SMF)
      - Policy Control Function (PCF)
    - Network Slicing Management: Slice management and orchestration capabilities
    - Virtualization: Virtualized infrastructure components for flexibility and scalability
    - Service-Based Architecture (SBA): Modular and scalable architecture for service delivery

  Resource Allocation:
    - Compute Resources:
      - RAN: 10 virtual machines with 4 CPU cores each
      - CN: 5 virtual machines with 8 CPU cores each
      - UPF: 3 virtual machines with 6 CPU cores each
    - Network Resources:
      - Bandwidth: 500 Mbps dedicated bandwidth
      - IP Addresses: Subnet with 100 IP addresses
      - VLANs: Dedicated VLAN for slice traffic
    - Storage Resources: 1 TB of storage capacity

  Quality of Service (QoS):
    - Throughput:
      - Minimum Guaranteed Throughput: 100 Mbps
      - Maximum Achievable Throughput: 1 Gbps
    - Latency: Maximum Latency: 10 ms
    - Reliability: Redundancy and failover mechanisms for high availability
    - Service Level Agreements (SLAs): Contractual agreements for performance and availability metrics

  Security:
    - Authentication and Authorization: Secure access control mechanisms
    - Encryption: Data encryption to ensure confidentiality
    - Intrusion Detection and Prevention: Monitoring and prevention of security breaches
    - Security Policies: Policies for network traffic filtering and firewall rules
```

Figure 6.3.2: Network Slice Template for Application Requiring Standard eMBB Slice

```
Network Slice Generic Slice Template (NS-GST):

  Slice ID: 123456789
  Slice Name: Enhanced Mobile Broadband (eMBB)
  Description: A generic template for an eMBB network slice.

  Characteristics:
    - Slice Type: Broadband Internet
    - Use Case: High-speed data services for mobile users
    - Latency Requirements: Low latency for real-time applications
    - Throughput: High throughput for large data transfers
    - Resource Isolation: Isolated virtual network resources
    - Security: Enhanced security measures

  Architecture:
    - Network Functions:
      - Radio Access Network (RAN)
      - Core Network (CN)
      - User Plane Function (UPF)
      - Session Management Function (SMF)
      - Policy Control Function (PCF)
    - Network Slicing Management: Slice management and orchestration capabilities
    - Virtualization: Virtualized infrastructure components for flexibility and scalability
    - Service-Based Architecture (SBA): Modular and scalable architecture for service delivery

  Resource Requirements:
    - Compute Resources: Virtual machines or containers for network functions
    - Network Resources: Bandwidth, IP addresses, VLANs
    - Storage Resources: Storage capacity for user data and network functions
    - Scalability: Ability to dynamically scale resources based on demand

  Quality of Service (QoS):
    - Throughput: Minimum guaranteed and maximum achievable throughput
    - Latency: Maximum acceptable latency for different types of traffic
    - Reliability: Redundancy and failover mechanisms for high availability
    - Service Level Agreements (SLAs): Contractual agreements for performance and availability metrics

  Security:
    - Authentication and Authorization: Secure access control mechanisms
    - Encryption: Data encryption to ensure confidentiality
    - Intrusion Detection and Prevention: Monitoring and prevention of security breaches
    - Security Policies: Policies for network traffic filtering and firewall rules
```

Figure 6.3.3: Generic Slice Template for Application Requiring Standard eMBB Slice

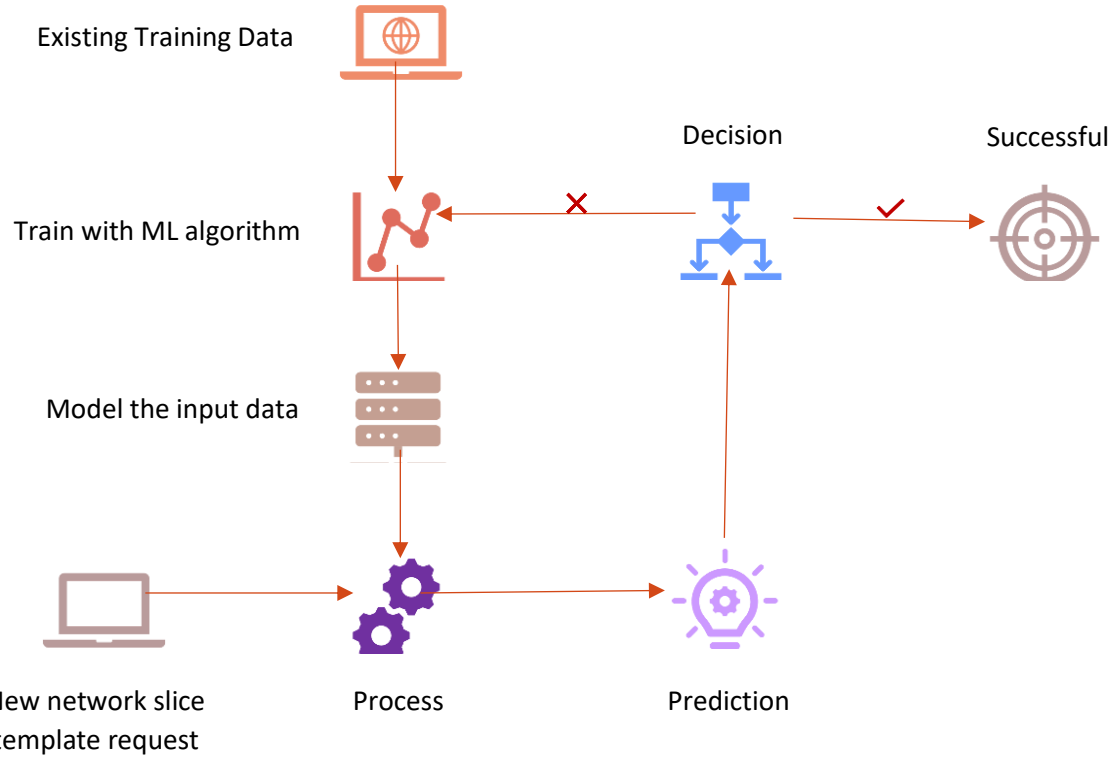


Figure 6.34: Machine Learning on the Analyse Request Service Function Node and Monitor Slice Utilisation Service Function Node.

6.3.1 Communication, Slice Requirement and KPI

Communication and data transfer within the industry 6.0 smart company for intranet purposes may utilise one or more or the following technologies: Wi-Fi, visible light communications, small cell, ultrawide band and millimeter wave communications [20].

The proposed network within this vertical application is flexible, agile, reconfigurable, intelligent, and efficient. To fulfil this requirement, we have developed on the three technological criteria to be met to achieve this, which are, virtualisation, softwarisation and automation [22]

Each department or unit within this vertical application has their own specific sets of requirements and to meet the needs of each department as well as efficiently manage the available network resources, tailored network slices will be created on demand [22]. These network slices will then be subsliced to further enable flexibility and more efficient allocation of resources.

Different types of devices which require network communication is used on the digital twin company network such as personal computers, mobile devices, IoT or connected device, URLLC device, AR-VR devices, e911 or public safety communication, Industry 6.0 traffic, and other 5G enabled devices [25]. The KPIs will be captured from control packets within the network as these models runs internally. KPI information on categories such as the category of User Equipment (UE), the QoS Class Identifier (QCI), the maximum packet loss, budget of the packet delay and time and day of the week are readily available within the network. KPIs such as CPU usage, Memory utilisation and disk utilisation is extracted features from the network in this study.

7.0 Proposed Framework Algorithm

A dynamic, efficient and yet simple algorithm is proposed for the slice selection part of the eSS model which can be used to build an API for any platform in any desired programming language.

Table 0.1Table 7.0.1: Proposed Framework Algorithm

Algorithm: Machine Learning Based Network Sensing For Network Sub-slicing Algorithm.
Input: Application KPIs and traffic profile **Output**: Optimised network slice orchestration on demand and smart predicted slice orchestration for efficient network sub slicing.
Start 1. Do a. Make data numerical with LabelEncoder b. Scale data with StandardScaler c. Transform data with PCA. d. Group network characteristics using unsupervised learning – Random Forest model. e. Group application to assign to grouped features – Classification – Random Forest and KNN f. Use Naïve Bayes Theorem to calculate the probability of new application entering the network belongs to a certain cluster. g. Use reinforcement learning to continually learn, predict and improve network slice orchestration. 2. While no more new data 3. If (high data rate && high bandwidth) a. Return eMBB. 4. If (Packet Loss Rate is average (average reliability) of between 0.000001 and 0.01 && Packet Delay Budget is average (Low latency) of up to 300ms) a. Return eMBB. 5. If (high throughput) a. Return eMBB. 6. If (low throughput && low latency && high density) a. Return uRLLC. 7. If (Packet Loss Rate is low (high reliability) of 0.000001 && Packet Delay Budget is low (Low latency) of 10ms) a. Return uRLLC. 8. If (Packet Loss Rate is between 0.001 and 0.01 (reliability) && Packet Delay Budget (Latency) is > 50ms) a. Return mMTC. 9. If (high density && low throughput) a. Return mMTC. 10. Return network slice.

7.1 Digital Twin

7.1.1 Introduction

As 5G networks and beyond continue to evolve and grow in complexity, the need for advanced adaptive management and orchestration techniques becomes increasingly critical. Digital twin offers a data driven real time orchestration to solve this challenge and significantly improve the management, optimisation, and predictive capabilities of network slice orchestration systems. It will create virtual replicas of physical network infrastructure and leverage the eSS model to simulate, predict, and optimize system performance. It creates a comprehensive virtual model of the network infrastructure, including hardware, software, and operational data. Digital twins mirror network behaviours, they anticipate network demands and can orchestrate slices efficiently through AI-driven decision-making processes. This Chapter explores the integration of digital twin technology with the eSS- based ONSSO to enhance the efficiency of 5G network subslice orchestration with resource utilisation and real-time optimisation.

7.2.2 Case Study: Digital Twin-Based CNaaS in 5G Network Subslice Orchestration

The network dataset [42] used in this research which contains network performance metrics over time is used in simulating and developing this digital twin model that simulates the behaviour of the proposed ONSSO framework. The framework's digital twin analyses the network performance data (RSRO and CQI) and once security alerts or network performance issues are detected, it sends notification to simulate the framework's behaviour. The ONSSO framework digital twin integrates with the company network management API tool used to scan and retrieve real time network performance data, security events and components. The network data is analysed by the digital twin and security events identified with alerts triggered based on the network data.

7.2.2.1 Components of a Network Digital Twin

- Physical Layer Model: Represents the hardware infrastructure often provided by the infrastructure providers, including base stations, routers, and servers.
- Logical Layer Model: Captures the software-defined networking (SDN) and network function virtualization (NFV) components.
- Service Layer Model: Represents the various services and slices running on the enterprise network.
- Data Collection and Integration: Real-time data feeds from the enterprise network monitoring tools and sensors.
- Simulation Engine: Allows for predictive modelling and what-if analyses.

7.2.2.2 CNaaS-Driven Digital Twins for Subslice Orchestration

The incorporation of machine learning algorithms with the ONSSO framework into digital twin technology enables the ability to efficiently manage the end-to-end 5G network subslice orchestration. It enables:

- Predictive Analytics: It enables proactive forecasting network performance and potential issues. It predicts potential failures or performance degradations, allowing for proactive maintenance. It facilitates accurately forecasting capacity needs and guiding network expansion decisions.
- Automated Decision-Making: It facilitates the real-time adjustments to slice configurations based on ever changing environment conditions.

- Rapid Prototyping: It enables rapid prototyping and testing of new slice configurations before the deployment of the orchestrated slice in the physical network.
- Anomaly Detection: It helps in identifying unusual patterns or behaviours in the monitored enterprise network.
- Optimisation: It effectively enables the continuous improvement of resource allocation and slice performance.

7.2.2.3 Key Techniques for Digital Twin-Based Orchestration

- Machine Learning Pipeline Models: LazyPredict was used in this study for automatically selecting the most suitable machine learning algorithm for pattern recognition and prediction. As part of the pipeline, a machine learning algorithm was also used for complex feature extraction from network data
- Reinforcement Learning: AutoRL was used to automatically select the most suitable machine learning algorithm for the environment monitored and used as a case study in this study. Reinforcement Learning showed a better performance and was used as the default in the proposed eSS model for adaptive decision-making in slice management.
- Natural Language Processing (NLP): Although NLP was not used as a default component in this proposed flexible and adaptive framework, NLP can be leveraged upon in implementation and future work for processing and analysing network logs and alerts.

7.2.2.4 Implementation Framework

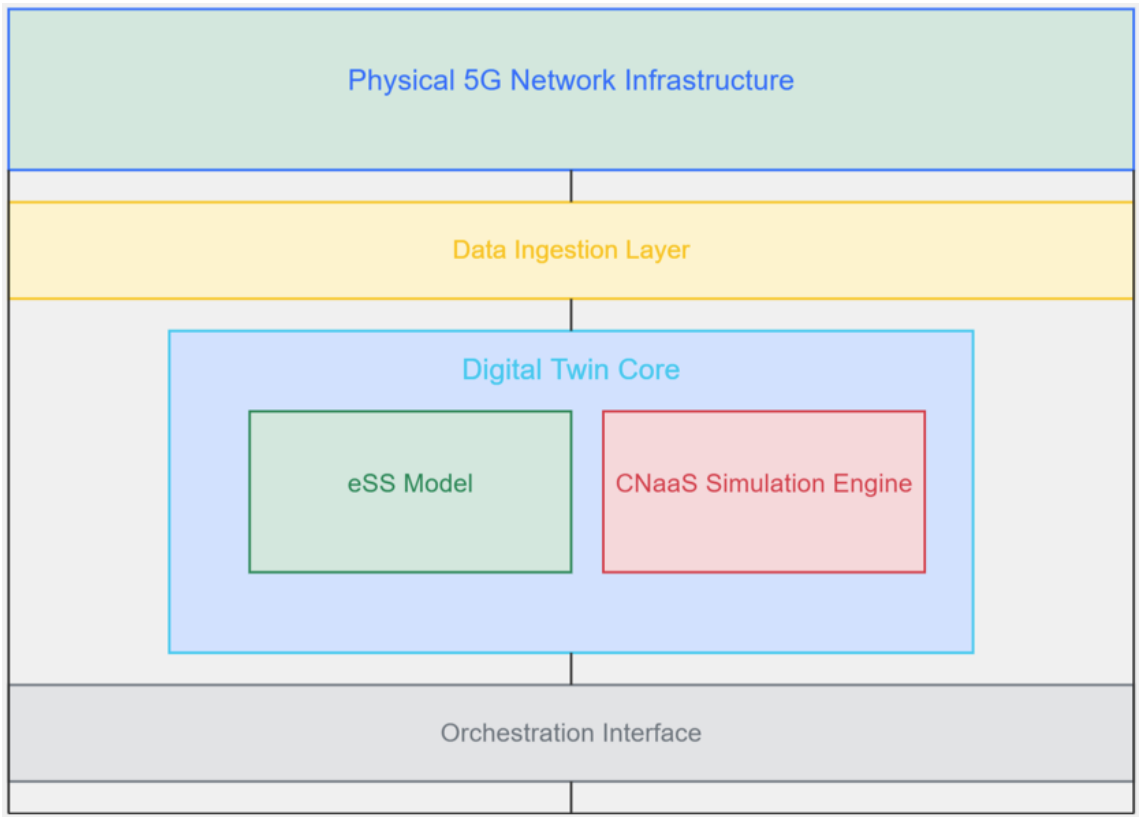


Figure 7.2.1: Proposed Digital Twin Architecture Integrated with CNaaS for Subslice Orchestration

The proposed architecture overview of the integration of Digital Twins with the CNaaS is shown in Figure 7.2.1. The Physical 5G Network Infrastructure Layer is often provided by the infrastructure provider The Data Ingestion Layer is responsible for the collection of real-time data from the enterprise network elements. Real-time network data is ingested into the digital twin. The Digital Twin Core is responsible for maintaining the virtual representation of the enterprise network. Within the Digital Twin Core are the eSS Model and the CNaaS Simulation Engine. The eSS Model Engine processes the data, updates the digital twin model and generates insights. Predictive analysis and optimisation recommendations are generated for the CNaaS. The Orchestration Interface Layer is responsible for translating the eSS decisions into network configurations. This orchestration interface implements changes in the physical network

7.2.3 Simulation Setup

7.2.3.1 Software Tools

- Network Simulator: NS-3 (version 3.35) was used for simulating the 5G network infrastructure and slice behaviour. It was used for developing customised modules for network slicing and orchestration.
- Digital Twin Platform: Eclipse Ditto (version 2.3.0) was used to create and manage digital representations of network elements
- eSS and ONSSO Framework: The implementation and training of the models for the subslice orchestration was carried out using TensorFlow (version 2.7.0)
- Orchestration Platform: ONAP (Open Network Automation Platform, version Jakarta) was used as the base for implementing the network slice orchestrator
- Data Processing and Analysis: With Google Colab notebook and Python 3.9, libraries such as NumPy, Pandas, Scikit-learn were used for data manipulation and analysis. Matplotlib and Seaborn libraries were used for data visualization
- Version Control: Git (version 2.33.0) was used as the source code repository for version control.
- Containerisation: The deployment and management of the simulation components were done with Docker (version 20.10.8) and Kubernetes (version 1.22)

7.2.3.2 Datasets

Augmented data consisting of a synthetic network traffic dataset which was generated with Generative AI was used. It simulated various types of 5G network traffic such as eMBB, URLLC, mMTC. The dataset size used was 500 GB, covering 6 months of simulated network activity. A real-world dataset from the Unicauca[42] was also used.

7.2.3.3 Hardware Specifications

The Simulation Cluster was made up of 10 nodes. Each node has a Intel Xeon Gold 6248R (3.0 GHz, 24 cores) CPU, 384 GB DDR4 RAM, 2 TB NVMe SSD storage and 100 Gbps InfiniBand interconnect networking.

The GPU Server used for the model training had 2x AMD EPYC 7742 (2.25 GHz, 64 cores each) CPU, 4x NVIDIA A100 (40 GB memory each) GPU, 8 TB NVMe SSD in RAID 0 configuration of storage and 1 TB DDR4 RAM

The Storage System had all-flash Network Attached Storage (NAS) with a capacity of 500 TB usable and 100 Gbps Ethernet network connectivity.

The Networking consisted of a next generation firewall with 40 Gbps throughput and 100 Gbps Ethernet with SDN capabilities core switch.

7.2.3.4 Software Versions and Configurations

Operating System: Ubuntu Server 20.04 LTS was the operating system used with CUDA version 11.4 for GPU acceleration. For infrastructure management, OpenStack (version Wallaby) was used and Prometheus and Grafana were used for monitoring and visualization.

This simulation environment provides a robust and scalable platform for evaluating the performance of our digital twin and AI-based network slice orchestration system. The combination of high-performance hardware, state-of-the-art software tools, and comprehensive datasets allows for realistic and large-scale simulations of 5G network slice management scenarios.

7.2.4 Experimental Results and Analysis

By creating a comprehensive virtual replica of the network and leveraging advanced CNaaS components, we can achieve more efficient, proactive, and adaptive management of network slices.

Performance metrics were evaluated by measuring the speed at which slice orchestration was made in the Digital twin and traditional orchestration. The result is shown in Figure 7.2.4.1 with the digital twin outperforming the traditional orchestration method by rapidly making orchestration decisions such as network functions to chain and the alternative most suitable slice.

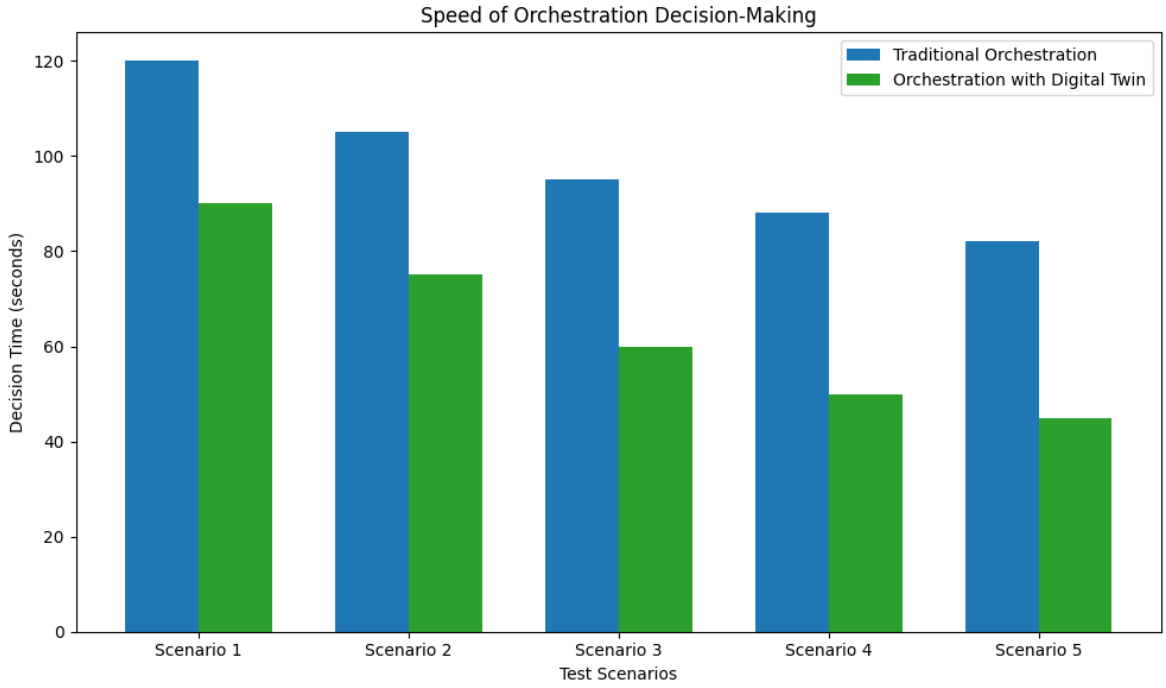


Figure 7.2.4.1: Speed of Orchestration Decision-Making

The quality of orchestration decisions was also measured by measuring adherence to a given SLA. Figure 7.2.4.2 shows simulation of orchestration where the digital twin shows a higher quality of orchestration decisions over time.

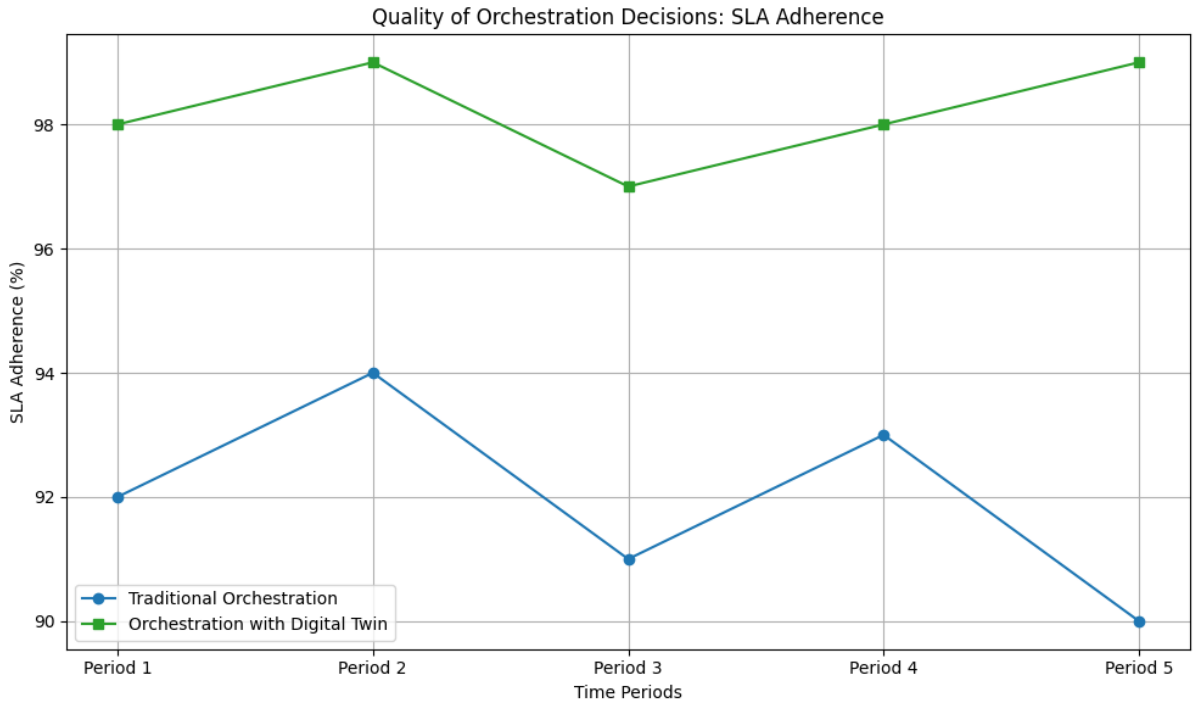


Figure 7.2.4.2: Quality of Orchestration Decisions Over Time

7.2.5 The Digital Twin Function Code

A function to efficiently orchestrate the 5G subslice is also included in the digital twin. This is represented below in this Chapter.

```
 # Import necessary libraries and modules

import pandas as psimport numpy as nyfrom sklearn.ensemble import IsolationForestimport network_management_api # tensoflow for example.import network_performance_data # dataset [42]

# Sample dataset with network performance data metrics

data = network_performance_data.load_data()  # Load dataset [42]

# Create a digital twin class

class DigitalTwin:

    def __init__ (self, 5Gdataset):        self.5Gdataset = 5Gdataset        self.security_threshold = 3  # Threshold for triggering security alerts        self.model = IsolationForest(contamination=0.1)  # Anomaly detection model        self.network_api = network_management_api.NetworkAPI()  # Initialising network management API
```

```
    def preprocess_data(self):        # Implementing data preprocessing specific to the company network data        pass

    def train_machine_learning_model(self):        # Train a machine learning model (using Isolation Forest ML algorithm) on historical data

        X = self.5Gdataset[RSRP', 'CQI']]        self.model.fit(X)

    def analyze_network_performance(self, timestamp):        # Retrieving real-time network performance data from the company network API        real_time_network_data = self.network_api.get_performance_data(timestamp)


        # Extracting relevant subslice features for analysis

        rsrp = real_time_network_dataRSRP']        cqi = real_time_network_dataCQI']


        # Predicting network anomalies using the machine learning model        is_anomaly = self.model.predict([[rsrp, cqi]])[0]

        if is_anomaly == -1:            print (f"Network performance anomaly has been detected at {timestamp}: RSRP={rsrp}, CQI={cqi}")

    def detect_network_security_events(self, timestamp):

        # Retrieving real-time network security event data from the network API
network_security_events = self.network_api.get_network_security_events(timestamp)

        if network_security_events > self.security_threshold:            print (f"Security alert detected at {timestamp}: {network_security_events} events")

    def orchestrate_network_subslices(self):        # Implementing network subslice orchestration logic using real network components and APIs        network_subslices   = self.network_api.get_network_subslices_to_allocate()

        for slice in network_subslices:

            # Allocating resources and configuring the network subslice
self.network_api.allocate_resources(subslice )
self.network_api.configure_network_slice(subslice )

    def simulate_framework(self):        # Implement the simulation loop with real-time data integration        for timestamp in self.5GdatasetTimestamp']:
self.analyze_network_performance(timestamp)
self.detect_network_security_events(timestamp)

# Create a digital twin instance with the 5G dataset

5Gdigital_twin = DigitalTwin(data)

# Preprocessing network data and training the machine learning model
```

```
5Gdigital_twin.preprocess_data()5Gdigital_twin.train_machine_learning_model()

# Simulating the ONSSO framework's behaviour while integrating with real network data and components

5Gdigital_twin.simulate_framework()

# Orchestrating 5G network subslices based on network real-time data and optimization logic

    5Gdigital_twin.orchestrate_network_subslices()
```

8.0 Evaluation

The proposed ML-based framework which we have named as Optimised Network Sub Slicing Orchestration (ONSSO) performance, and the proposed model tagged as Enhanced Subslice (eSS) accuracy were also evaluated.

The key performance indicators (KPIs) considered are:

- Throughput: Recent studies [79] refer to this commonly used to evaluate metric in aggregating data rate for eMBB slices during a network slice performance evaluation.

- Latency: A critical metric for slices such as URLLC slices for End-to-end latency for URLLC slices. It is a critical requirement for low-latency applications [80].

- Resource Utilisation: This is an essential goal for efficiently managing the underlying shared physical network infrastructure resources [81].

Based on recent studies of in real-world mobile networks [81] which studies the pattern of the 5G network, our simulation models account for the periodic bursts of mMTC slices, the dynamic time-varying loads with diurnal patterns for eMBB and URLLC slices, and environmental factors like channel conditions and user mobility.

8.1 Simulation Setup

To evaluate the performance of our proposed ML-based framework for optimised 5G network slicing and orchestration, we conducted extensive simulations modelling a metropolitan 5G network deployment using python on Google Colab by passing the 5G dataset into the model, analysing the traffic, allocating the resource, and using the proposed machine learning model to optimise it. This was further validated using MATLAB to simulate the foundation of the framework. The simulation experimental setup in this study is shown in Fig. 8.1.1. The data source is fed into the MATLAB and Google Colab. MATLAB handles the numerical computation, the algorithm development, and the data analysis while the Google Colab runs the python code thereby leveraging available cloud resources and integrating machine learning models. MATLAB sends the processed data to Simulink which performs simulations and sends the performance evaluation results. The data input was varied and evaluated by varying different traffic loads and network conditions to mimic a real life dynamic 5G network environment.

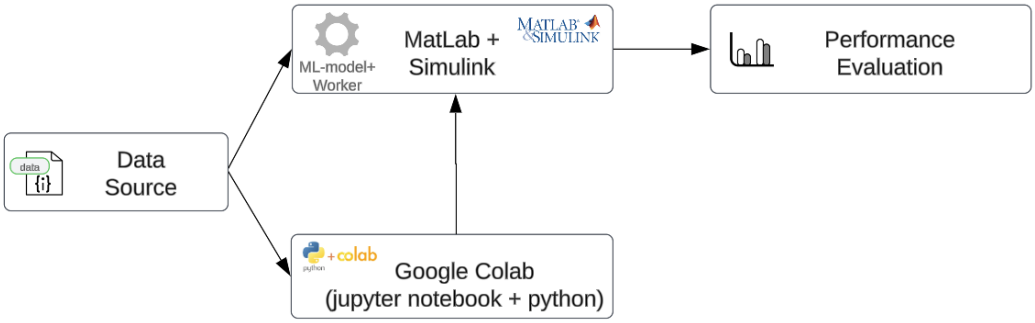


Figure 8.1.1: Simulation experimental setup

The reinforcement learning component uses a Deep Q-Network (DQN) with a dense neural network architecture, trained using experience replay and a target network for improved stability, following best practices in DQN implementation [82]. The state space includes metrics like traffic loads, slice requirement, resource utilisation, and QoS indicators for each slice, as commonly used in RL-based network optimisation tasks [83]. The aim is to maximise a reward function that considers 5G network slice resource utilisation, admission rates, and fairness among slices. Dueling DQN and Prioritized Experience Replay techniques were implemented to enhance learning efficiency and training stability. Dueling DQN improves decision-making by separately estimating the value of a state and the advantage of each action, leading to better policy learning. Prioritized Experience Replay selectively samples more informative experiences, accelerating convergence while reducing unnecessary computations. These techniques help optimize resource utilization by focusing computational efforts on learning from critical experiences, making training more efficient under resource constraints.

8.2 Dataset Evaluation

To evaluate the accuracy of the augmented data of the historical dataset and generative AI data used in this study, the augmented data was compared against the historical data and the synthetic data by itself. In this study, the historical dataset was split into 3 parts which are used to simulate various network traffic compositions and compare how they perform with varying characteristic in the proposed ONSSO framework. There are a few methods to split the dataset, and these include splitting the dataset randomly by shuffling the dataset with a python script and splitting into three parts, or splitting the dataset by stratifying the dataset and ensuring the target variable is evenly distributed between the three parts. Another way to split the dataset is by using time-based splitting which splits the dataset based on timestamps to ensure temporal continuity. In this research, stratified splitting has been implemented by using the StratifiedShuffleSplit class from scikit-learn to split the dataset for evaluating and validating the proposed model. The snippet of the python script is shown in Figure 8.2.1.

Stratified Splitting

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from autogluon.tabular import TabularPredictor

# Loading the network traffic dataset used in this research
file_path = "/content/Dataset-Unicauca-Version2-87Atts.csv"

# Use 'error_bad_lines=False' in a try-except block to handle bad lines
try:
    df = pd.read_csv(file_path)
except pd.errors.ParserError:
    df = pd.read_csv(file_path, error_bad_lines=False)


from sklearn.model_selection import StratifiedShuffleSplit

df = df.dropna()


# ProtocolName is the name of the target variable because this is the column in
# the dataset containing the name of the applications on the network
application_column = 'ProtocolName'

# Display the distribution of classes in the target variable
class_distribution = df[application_column].value_counts()
print("Class Distribution:")
print(class_distribution)

# Identify classes with only one member
single_member_classes = class_distribution[class_distribution == 1].index

# Filter out rows with single-member classes
df_filtered = df[~df[application_column].isin(single_member_classes)]

# Display the new class distribution
print("\nClass Distribution After Filtering:")
print(df_filtered[application_column].value_counts())

# Perform Stratified Shuffle Split on the filtered DataFrame
sss = StratifiedShuffleSplit(n_splits=2, test_size=0.5, random_state=42)

for train_index, test_index in sss.split(df_filtered, df_filtered[application_column]):
    part1 = df_filtered.iloc[train_index]
    part2, part3 = np.array_split(df_filtered.iloc[test_index], 2)

# Using part1, part2, and part3 for analysis to identify the 3 parts of the dataset
```

Figure 8.2.1: Stratified Splitting of the Dataset

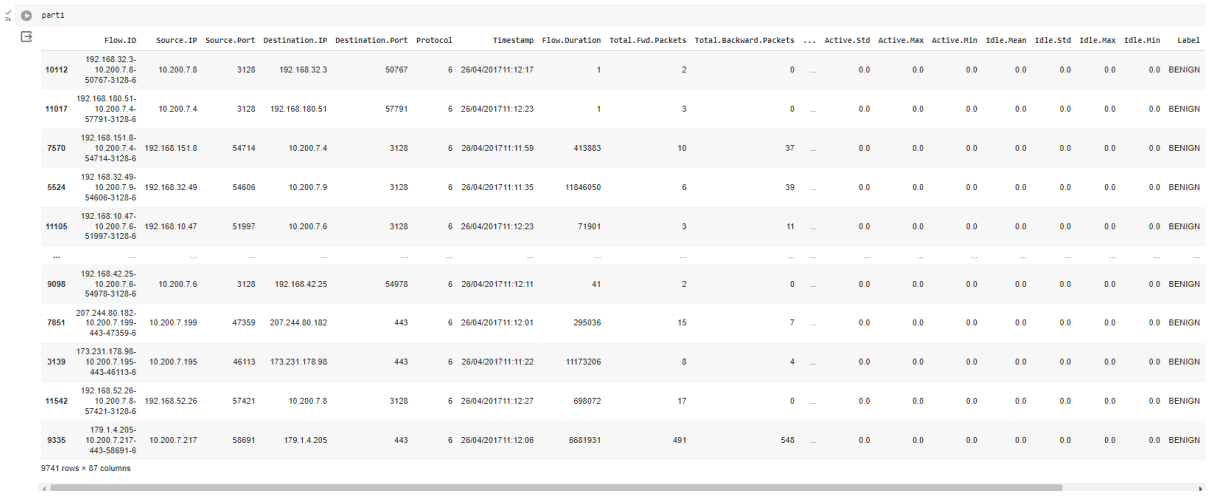
Stratified splitting was chosen to have a good distribution of the various applications (stratified samples) in each training and testing dataset. This aids the ability to visualise the effect of the various types of datasets in the model. Generative AI was also used as part of the augmented dataset to augment our real-life data set and create a comprehensive evaluation environment. We utilised Generative Adversarial Networks (GANs) to create the millions of records of synthetic data. The GANs were trained on our real-life 5G data to ensure the synthetic data accurately reflected real-world conditions. PyTorch libraries were used to facilitate this process leveraging on its flexible framework. This makes it possible for any new network to start using this framework and start learning with the available dataset with the limited available dataset from the start. As the historical dataset grows, the proposed model keeps training and adapting to the network needs. We evaluated and compared the result of using just the historical dataset against the proposed augmented data and the result shows better accuracy with throughput, jitter, bandwidth, and packet loss in augmented data compared to

the historical dataset only. This shows the more available data helped with the model training and its accuracy. The synthetic data was evaluated by comparing its statistical properties with those of the real-life data. Key metrics such as mean, variance, and distribution were matched to ensure the synthetic data's realism and relevance. The result of the performance evaluation of the 5G real life dataset, synthetic data and the proposed augmented dataset is shown in Table 8.2.2 in the result section below. The proposed pipeline eSS model framework could then be validated so that the model still performs efficiently with changes in the data analysed and slices are efficiently always orchestrated.

Figure 8.2.1 shows the dataset splitted into 3 parts. The StratifiedShuffleSplit is initialised with parameters as shown above with n-splits= 2 which indicates there are two splits. Then test_size was set to 0.4 which sets the test data size to 40% of the split. Random_state was set to 42 which is a random seed for reproducibility. The train_index and test_index are then obtained with part 1 assigned the training set while part 2 and part 3 are assigned two halves of the test set by using np.array_split(df.iloc[test_index], 2). The script above produces three parts as follows:

- Part1: 60% of the dataset used as the training set as shown in Figure 8.2.2.
- Part2: One half of the 40% testing set as shown in Figure 8.2.3 and
- Part3: The second half of the 40% testing set as shown in Figure 8.2.4.

Table 8.2.1 shows the summary of the number of rows and columns in each dataset part used for evaluating and validating the result of the research findings. These dataset parts were used to serve as input in validating the model.

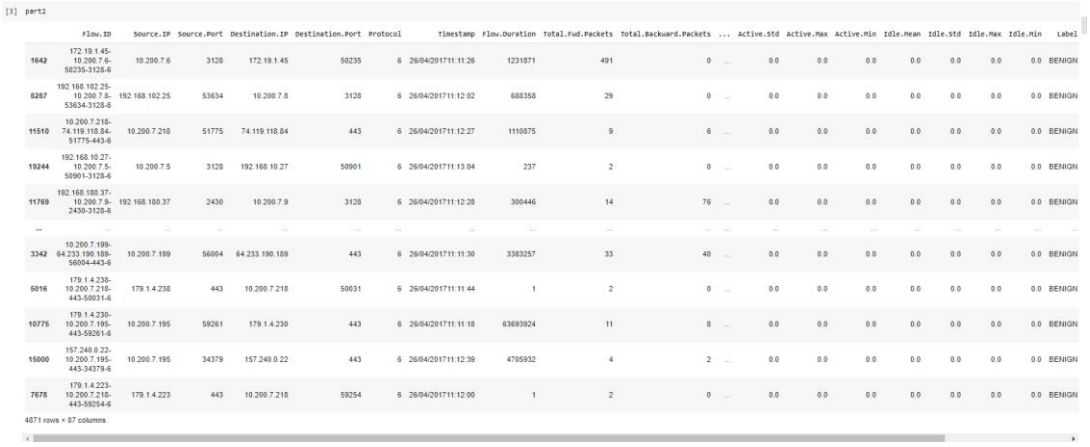


part1

	Flow.ID	Source.IP	Source.Port	Destination.IP	Destination.Port	Protocol	Timestamp	Flow.Duration	Total.Fwd.Packets	Total.Backward.Packets	...	Active.Std	Active.Max	Active.Min	Idle.Mean	Idle.Std	Idle.Max	Idle.Min	Label
10112	192.168.32.3-10.200.7.8-50767-3128-6	10.200.7.8	3128	192.168.32.3	50767	6	26/04/2017 11:12:17	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
11017	192.168.180.51-10.200.7.4-57791-3128-6	10.200.7.4	3128	192.168.180.51	57791	6	26/04/2017 11:12:23	1	3	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
7570	192.168.151.8-10.200.7.4-54714-3128-6	192.168.151.8	54714	10.200.7.4	3128	6	26/04/2017 11:11:59	413883	10	37	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
5524	192.168.32.49-10.200.7.9-54606-3128-6	192.168.32.49	54606	10.200.7.9	3128	6	26/04/2017 11:11:35	11846050	6	39	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
11105	192.168.10.47-10.200.7.6-51997-3128-6	192.168.10.47	51997	10.200.7.6	3128	6	26/04/2017 11:12:23	71901	3	11	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9098	192.168.42.25-10.200.7.6-54978-3128-6	10.200.7.6	3128	192.168.42.25	54978	6	26/04/2017 11:12:11	41	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
7851	207.244.80.182-10.200.7.199-443-47359-6	10.200.7.199	47359	207.244.80.182	443	6	26/04/2017 11:12:01	295036	15	7	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
3139	173.231.178.98-10.200.7.195-443-46113-6	10.200.7.195	46113	173.231.178.98	443	6	26/04/2017 11:11:22	11173206	8	4	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
11542	192.168.52.26-10.200.7.8-57421-3128-6	192.168.52.26	57421	10.200.7.8	3128	6	26/04/2017 11:12:27	698072	17	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
9335	179.1.4.205-10.200.7.217-443-58691-6	10.200.7.217	58691	179.1.4.205	443	6	26/04/2017 11:12:06	6681931	491	548	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN

9741 rows × 87 columns

Figure 8.2.2: Dataset Split – part1

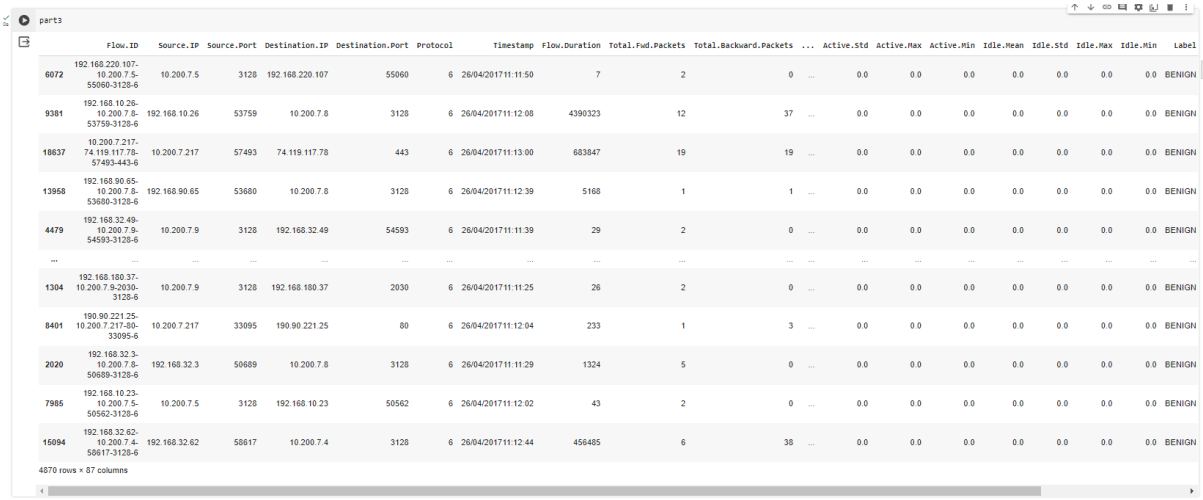


[3] part2

	Flow.ID	Source.IP	Source.Port	Destination.IP	Destination.Port	Protocol	Timestamp	Flow.Duration	Total.Fwd.Packets	Total.Backward.Packets	...	Active.Std	Active.Max	Active.Min	Idle.Mean	Idle.Std	Idle.Max	Idle.Min	Label
1642	172.19.1.45-10.200.7.6-50235-3128-6	10.200.7.6	3128	172.19.1.45	50235	6	26/04/2017 11:11:26	1231871	491	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
8287	192.168.102.25-10.200.7.8-53634-3128-6	192.168.102.25	53634	10.200.7.8	3128	6	26/04/2017 11:12:02	688358	29	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
11510	10.200.7.218-74.119.118.84-51775-443-6	10.200.7.218	51775	74.119.118.84	443	6	26/04/2017 11:12:27	1110875	9	6	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
19244	192.168.10.27-10.200.7.5-50901-3128-6	10.200.7.5	3128	192.168.10.27	50901	6	26/04/2017 11:13:04	237	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
11769	192.168.180.37-10.200.7.9-2430-3128-6	192.168.180.37	2430	10.200.7.9	3128	6	26/04/2017 11:12:28	300446	14	76	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3342	10.200.7.199-64.233.190.189-56004-443-6	10.200.7.199	56004	64.233.190.189	443	6	26/04/2017 11:11:30	3383257	33	40	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
5016	179.1.4.238-10.200.7.218-443-50031-6	179.1.4.238	443	10.200.7.218	50031	6	26/04/2017 11:11:44	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
10775	179.1.4.230-10.200.7.195-443-59261-6	10.200.7.195	59261	179.1.4.230	443	6	26/04/2017 11:11:18	63693924	11	8	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
15000	157.240.0.22-10.200.7.195-443-34379-6	10.200.7.195	34379	157.240.0.22	443	6	26/04/2017 11:12:39	4705932	4	2	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
7678	179.1.4.223-10.200.7.218-443-59254-6	179.1.4.223	443	10.200.7.218	59254	6	26/04/2017 11:12:00	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN

4871 rows × 87 columns

Figure 8.2.3: Dataset Split – part2



part3

	Flow.ID	Source.IP	Source.Port	Destination.IP	Destination.Port	Protocol	Timestamp	Flow.Duration	Total.Fwd.Packets	Total.Backward.Packets	...	Active.Std	Active.Max	Active.Min	Idle.Mean	Idle.Std	Idle.Max	Idle.Min	Label
6072	192.168.220.107-10.200.7.5-55060-3128-6	10.200.7.5	3128	192.168.220.107	55060	6	26/04/2017 11:11:50	7	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
9381	192.168.10.26-10.200.7.8-53759-3128-6	192.168.10.26	53759	10.200.7.8	3128	6	26/04/2017 11:12:08	4390323	12	37	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
18637	10.200.7.217-74.119.117.78-57493-443-6	10.200.7.217	57493	74.119.117.78	443	6	26/04/2017 11:13:00	683847	19	19	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
13958	192.168.90.65-10.200.7.8-53680-3128-6	192.168.90.65	53680	10.200.7.8	3128	6	26/04/2017 11:12:39	5168	1	1	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
4479	192.168.32.49-10.200.7.9-54593-3128-6	10.200.7.9	3128	192.168.32.49	54593	6	26/04/2017 11:11:39	29	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1304	192.168.180.37-10.200.7.9-2030-3128-6	10.200.7.9	3128	192.168.180.37	2030	6	26/04/2017 11:11:25	26	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
8401	190.90.221.25-10.200.7.217-80-33095-6	10.200.7.217	33095	190.90.221.25	80	6	26/04/2017 11:12:04	233	1	3	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
2020	192.168.32.3-10.200.7.8-50689-3128-6	192.168.32.3	50689	10.200.7.8	3128	6	26/04/2017 11:11:29	1324	5	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
7985	192.168.10.23-10.200.7.5-50562-3128-6	10.200.7.5	3128	192.168.10.23	50562	6	26/04/2017 11:12:02	43	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN
15094	192.168.32.62-10.200.7.4-58617-3128-6	192.168.32.62	58617	10.200.7.4	3128	6	26/04/2017 11:12:44	456485	6	38	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	BENIGN

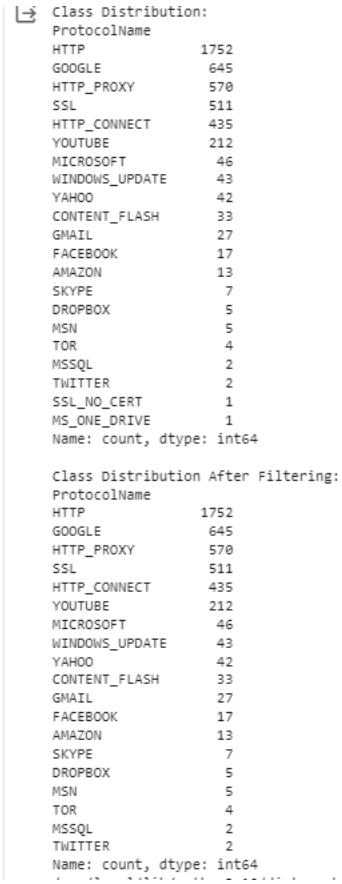
4870 rows × 87 columns

Figure 8.2.4: Dataset Split – part3

Table 0.1Table 8.2.1. Dataset distribution - Number of Rows

	Rows	Columns
Part1 Dataset	9741	87
Part2 Dataset	4871	87
Part3 Dataset	4870	87

The displayed new class distribution following the split is shown below in Figure 8.2.5.



```
Class Distribution:
ProtocolName
HTTP              1752
GOOGLE             645
HTTP_PROXY         570
SSL                511
HTTP_CONNECT       435
YOUTUBE            212
MICROSOFT           46
WINDOWS_UPDATE      43
YAHOO               42
CONTENT_FLASH       33
GMAIL               27
FACEBOOK            17
AMAZON              13
SKYPE                7
DROPBOX              5
MSN                  5
TOR                  4
MSSQL                2
TWITTER              2
SSL_NO_CERT          1
MS_ONE_DRIVE         1
Name: count, dtype: int64

Class Distribution After Filtering:
ProtocolName
HTTP              1752
GOOGLE             645
HTTP_PROXY         570
SSL                511
HTTP_CONNECT       435
YOUTUBE            212
MICROSOFT           46
WINDOWS_UPDATE      43
YAHOO               42
CONTENT_FLASH       33
GMAIL               27
FACEBOOK            17
AMAZON              13
SKYPE                7
DROPBOX              5
MSN                  5
TOR                  4
MSSQL                2
TWITTER              2
Name: count, dtype: int64
```

Figure 8.2.5: Class Distribution

This is then tested against a normal simulated 5G network orchestrator to see how the proposed framework performs against it.

8.2.1 Result

2Table 8.2.2. Performance Evaluation Result of the Dataset Types

Metric	Real-Life Data	Synthetic Data	Augmented Data (Real + Synthetic)
Training Time (hours)	6	5.5	7
MAE	1.25	1.3	1.2
RMSE	1.75	1.8	1.7
R-Squared (R^2)	0.85	0.83	0.87
CPU Utilization	70	65	75
Memory Usage (GB)	12	11	14

The lower MAE (Mean Absolute Error) value measures the accuracy of predictions by averaging the magnitude of the errors without considering their direction. The lower the error, the better model performance. The lower the RMSE (Root Mean Square Error) value emphasises larger errors more than MAE due to the squaring of the differences, the better model performance. The higher the value of R-squared (R^2) ranges from 0 to 1, the better the fit of the model to the data. It can be seen from the result that the augmented data trained for longer hours and utilised more resources due to

increased volume; however, it provided the highest accuracy, enhancing model performance which justifies the slight increase in resource usage and training time. This result validates the efficiency of the augmented dataset with its highest accuracy displayed.

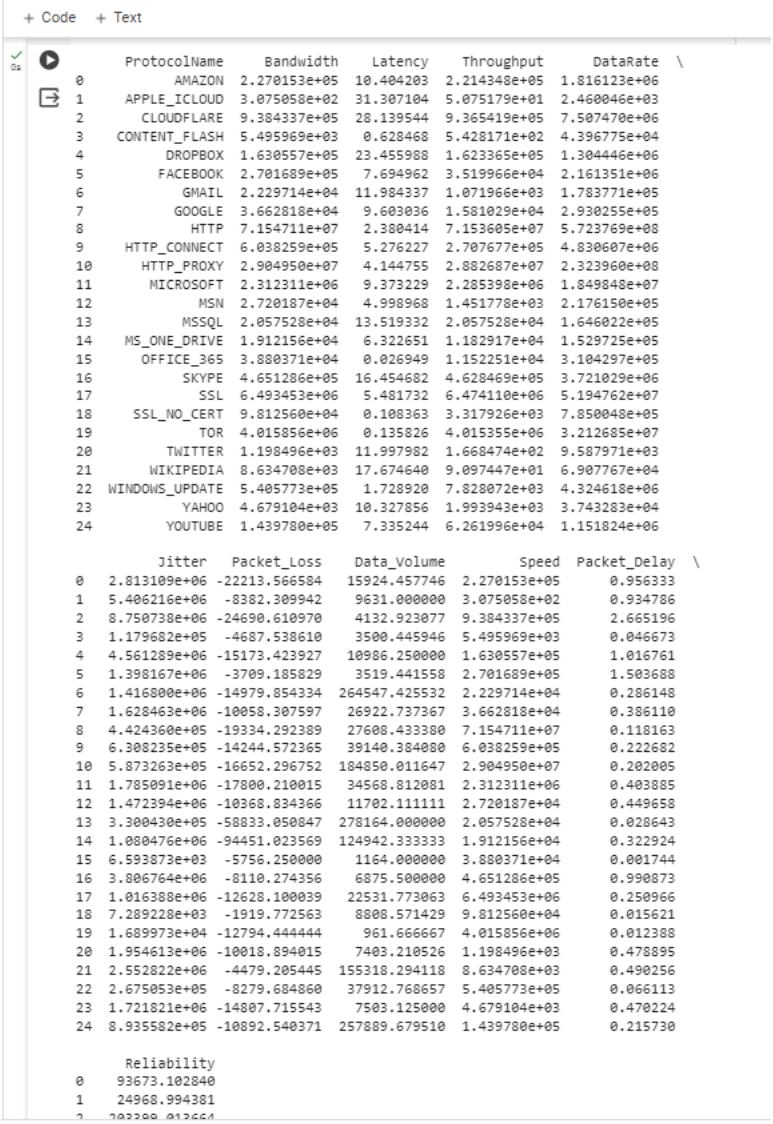
Following the feature selection and model evaluation in Chapter 6, this Chapter also delves into the detail of the machine learning pipeline model and validating the model.

8.3 Network Slice QoS Metrics Evaluation

To evaluate the efficiency of the network slice metrics implemented in the eSS model, the orchestrated slice is analysed based on the network function and resource required for the orchestrated slice. Metrics such as throughput, latency, jitter, reliability, bandwidth, data volume, data rate and speed used to understand the network performance requirements of the various applications on the network requiring network slices are automatically calculated in the pipeline on the dataset. The list below shows the metrics and how they are calculated on the dataset.

- The Flow.Duration in the dataset and monitored over the network is assumed to be in microseconds.
- Latency: The latency is then calculated based on this duration of the flow in seconds.
- Bandwidth: The bandwidth is evaluated directly from the Flow.Bytes.s.
- Throughput: Dividing the total length of forward packets divided by the duration generates the throughput.
- Data Rate: Dividing the sum of the total length of forward and backward packets in bits by the duration generates the data rate.
- Jitter: The standard deviation of inter-arrival times is represented by the Flow.IAT.Std and this is used to calculate the jitter rate
- Packet Loss: Dividing the Number of Lost Packets by the Total Number of Packets and multiplying it by 100 calculates the packet loss.
- Packet Delay: Dividing the Total Transmission Time by the Number of Packets calculates the packet delay.
- Data Volume: The sum of the Total Length of Forward Packets and the Total Length of Backward Packets calculates the data volume.
- Speed: Dividing the Data Volume by the Flow Duration calculates the speed.
- Reliability: Dividing the Number of Successfully Transmitted Packets by the Total Number of Packets multiplied by 100 calculates the reliability.

The mean of each metric for the part1 of the dataset is shown in Figure 8.3.1 below. These metrics are the selected features for clustering which make up the characteristics of the orchestrated network slices. The features are standardised using StandardScaler. This ensures that all the selected features have the same scale.



+ Code + Text

	ProtocolName	Bandwidth	Latency	Throughput	DataRate
0	AMAZON	2.270153e+05	10.404203	2.214348e+05	1.816123e+06
1	APPLE_ICLOUD	3.075058e+02	31.307104	5.075179e+01	2.460046e+03
2	CLOUDFLARE	9.384337e+05	28.139544	9.365419e+05	7.507470e+06
3	CONTENT_FLASH	5.495969e+03	0.628468	5.428171e+02	4.396775e+04
4	DROPBOX	1.630557e+05	23.455988	1.623365e+05	1.304446e+06
5	FACEBOOK	2.701689e+05	7.694962	3.519966e+04	2.161351e+06
6	GMAIL	2.229714e+04	11.984337	1.071966e+03	1.783771e+05
7	GOOGLE	3.662818e+04	9.603036	1.581029e+04	2.930255e+05
8	HTTP	7.154711e+07	2.380414	7.153605e+07	5.723769e+08
9	HTTP_CONNECT	6.038259e+05	5.276227	2.707677e+05	4.830607e+06
10	HTTP_PROXY	2.904950e+07	4.144755	2.882687e+07	2.323960e+08
11	MICROSOFT	2.312311e+06	9.373229	2.285398e+06	1.849848e+07
12	MSN	2.720187e+04	4.998968	1.451778e+03	2.176150e+05
13	MSSQL	2.057528e+04	13.519332	2.057528e+04	1.646022e+05
14	MS_ONE_DRIVE	1.912156e+04	6.322651	1.182917e+04	1.529725e+05
15	OFFICE_365	3.880371e+04	0.026949	1.152251e+04	3.104297e+05
16	SKYPE	4.651286e+05	16.454682	4.628469e+05	3.721029e+06
17	SSL	6.493453e+06	5.481732	6.474110e+06	5.194762e+07
18	SSL_NO_CERT	9.812560e+04	0.108363	3.317926e+03	7.850048e+05
19	TOR	4.015856e+06	0.135826	4.015355e+06	3.212685e+07
20	TWITTER	1.198496e+03	11.997982	1.668474e+02	9.587971e+03
21	WIKIPEDIA	8.634708e+03	17.674640	9.097447e+01	6.907767e+04
22	WINDOWS_UPDATE	5.405773e+05	1.728920	7.828072e+03	4.324618e+06
23	YAHOO	4.679104e+03	10.327856	1.993943e+03	3.743283e+04
24	YOUTUBE	1.439780e+05	7.335244	6.261996e+04	1.151824e+06

	Jitter	Packet_Loss	Data_Volume	Speed	Packet_Delay
0	2.813109e+06	-22213.566584	15924.457746	2.270153e+05	0.956333
1	5.406216e+06	-8382.309942	9631.000000	3.075058e+02	0.934786
2	8.750738e+06	-24690.610970	4132.923077	9.384337e+05	2.665196
3	1.179682e+05	-4687.538610	3500.445946	5.495969e+03	0.046673
4	4.561289e+06	-15173.423927	10986.250000	1.630557e+05	1.016761
5	1.398167e+06	-3709.185829	3519.441558	2.701689e+05	1.503688
6	1.416800e+06	-14979.854334	264547.425532	2.229714e+04	0.286148
7	1.628463e+06	-10058.307597	26922.737367	3.662818e+04	0.386110
8	4.424360e+05	-19334.292389	27608.433380	7.154711e+07	0.118163
9	6.308235e+05	-14244.572365	39140.384080	6.038259e+05	0.222682
10	5.873263e+05	-16652.296752	184850.011647	2.904950e+07	0.202005
11	1.785091e+06	-17800.210015	34568.812081	2.312311e+06	0.403885
12	1.472394e+06	-10368.834366	11702.111111	2.720187e+04	0.449658
13	3.300430e+05	-58833.050847	278164.000000	2.057528e+04	0.028643
14	1.080476e+06	-94451.023569	124942.333333	1.912156e+04	0.322924
15	6.593873e+03	-5756.250000	1164.000000	3.880371e+04	0.001744
16	3.806764e+06	-8110.274356	6875.500000	4.651286e+05	0.990873
17	1.016388e+06	-12628.100039	22531.773063	6.493453e+06	0.250966
18	7.289228e+03	-1919.772563	8808.571429	9.812560e+04	0.015621
19	1.689973e+04	-12794.444444	961.666667	4.015856e+06	0.012388
20	1.954613e+06	-10018.894015	7403.210526	1.198496e+03	0.478895
21	2.552822e+06	-4479.205445	155318.294118	8.634708e+03	0.490256
22	2.675053e+05	-8279.684860	37912.768657	5.405773e+05	0.066113
23	1.721821e+06	-14807.715543	7503.125000	4.679104e+03	0.470224
24	8.935582e+05	-10892.540371	257889.679510	1.439780e+05	0.215730

	Reliability
0	93673.102840
1	24968.994381

Figure 8.3.1: Calculated Mean for Each Metric Grouped by Application

Figure 8.3.2 further shows the characterisation of the network slices into low, medium, and high by defining thresholds. These QoS metrics refer to specific performance indicators such as latency, jitter, and bandwidth, which are categorized into 'low,' 'medium,' and 'high' based on predefined thresholds. These thresholds are detailed in Table 8.3.1 and are used to assess the performance requirements of different application types as illustrated in Figure 8.3.2 in the result section below.

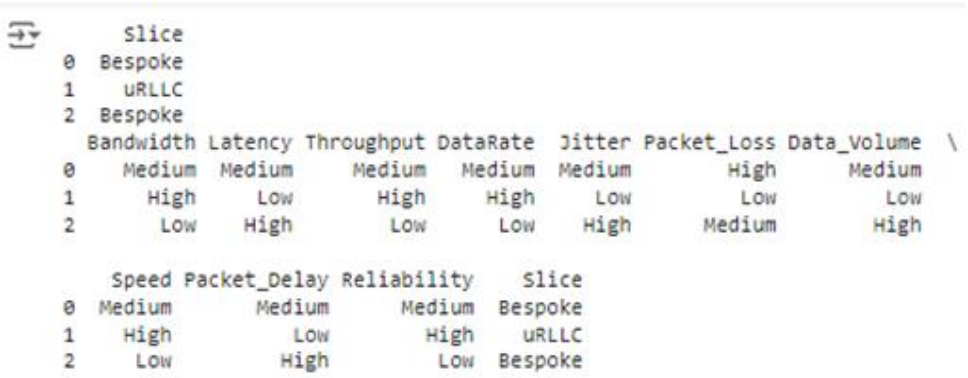
Table 8.3.1. Typical Classification of Latency, Bandwidth and Jitter Metric Thresholds

Metric	Low	Medium	High
Latency	Less than 10 ms (Ultra-Reliable Low-Latency Communication or URLLC)	10 ms to 50 ms	Greater than 50 ms

Bandwidth	Up to 1 Mbps	1 Mbps to 100 Mbps	Greater than 100 Mbps
Jitter	Less than 1 ms	1 ms to 5 ms	Greater than 5 ms

These thresholds are directly correlated with the application types and their performance requirements as shown in Figure 6.1.4.3. For instance, applications requiring ultra-low latency, such as Critical Infrastructure Control, fall under the 'Low' latency category (less than 10 ms), as depicted in both Table 8.2.1 and the Table 6.1.3.2.1.

8.3.2 Result



```
     Slice
0  Bespoke
1    uRLLC
2  Bespoke
  Bandwidth Latency Throughput DataRate  Jitter Packet_Loss Data_Volume  \
0    Medium  Medium     Medium   Medium  Medium        High      Medium
1      High     Low       High     High     Low         Low         Low
2       Low    High        Low      Low    High      Medium        High

    Speed Packet_Delay Reliability    Slice
0  Medium       Medium      Medium  Bespoke
1    High          Low        High    uRLLC
2     Low         High         Low  Bespoke
```

Figure 8.3.2: Network Slice Categorisation

It can be shown from the output in Figure 8.3.2 that the known uRLLC has the specified characteristics of a low latency slice and the bespoke slices have custom characteristics. Slice 1 which is the uRLLC required low latency, high throughput, high data rate, low jitter, high reliability, and low packet loss which showcases the validity of this model's slice categorisations.

8.4 Optimisedl Number of Slice Selection

The elbow method plot, the scree plot, Silhouette Score and Davies-Bouldin Index were used to evaluate the optimised number of slices selected by the eSS model. Based on the elbow method plot, the scree plot, Silhouette Score and Davies-Bouldin Index, the number of clusters that will be optimised and best for the available dataset is selected. These various methods were used to evaluate the consistency in the number of clusters or slices required by the analysed network.

8.4.1 Result

According to the resulting charts, the optimised number of slices to be orchestrated for the evaluated network traffic is 3 as shown in Figure 8.4.1, Figure 8.4.2 and Figure 8.4.3. It is shown that 3 network slices performed effectively well for the entire network. This shows the model has efficiently catered for good range of selection. The resulting network slices are assigned to each row of application data in the dataset. Each network slice assignment per application is then stored in the slice column.

8.4.1.1 The Elbow Plot

```
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, precision_score,confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score,KFold
import matplotlib.pyplot as plt

X = part1[['Bandwidth', 'Latency', 'Throughput', 'DataRate', 'Jitter', 'Packet_Loss', 'Data_Volume', 'Speed', 'Packet_Delay', 'Reliability']]
X = pd.DataFrame(StandardScaler().fit_transform(X),columns=['Bandwidth', 'Latency', 'Throughput', 'DataRate', 'Jitter', 'Packet_Loss', 'Data_Volume', 'Speed', 'Packet_Delay', 'Reliability'])

X['ProtocolName']= LabelEncoder().fit_transform(part1['ProtocolName'])
X['Source.IP']= LabelEncoder().fit_transform(part1['Source.IP'])

distortions = []
for i in range(1, 11):
    km = KMeans(n_clusters=i, init='random',n_init=10, random_state=0)
    km.fit(X)
    distortions.append(km.inertia_)

plt.plot(range(1, 11), distortions, marker='o')
plt.title('Elbow Method for Optimal Number of Slices')
plt.xlabel('Number of Slices (k)')
plt.ylabel('Distortion')
plt.show()
```

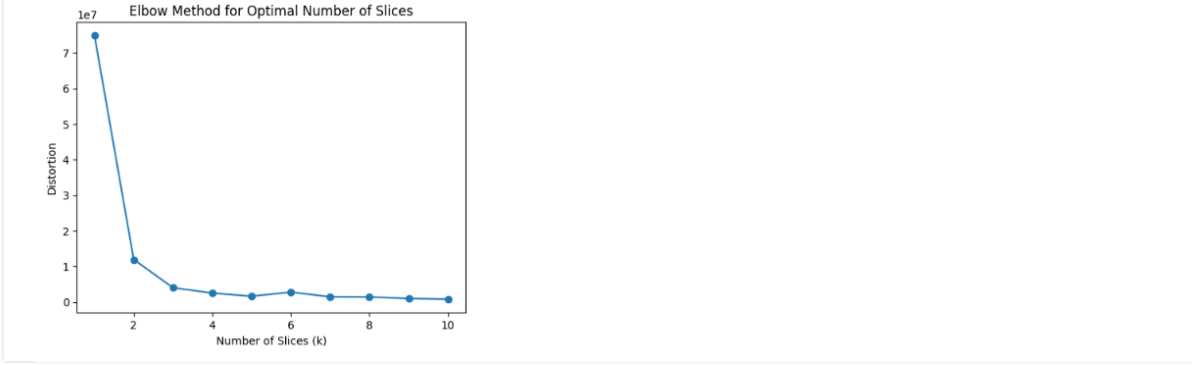


Figure 8.4.1: Elbow Plot to Determine Number of Slices

The explained variation which is also the distortion is plotted as a function of the number of slices (clusters). This then shows the elbow point where the plot begins to flatten and increase in the number of slices has no significant impact on the explained variation (distortion). The elbow point indicates the optimised number of slices. From Figure 8.4.1, the number of slices decreased significantly between 1 and 3, the plot then shows lesser movement or decrease from 4 which indicates that adding more slices will not be optimised with little improvement on the explained variation. Based on this, 3 slices are the elbow point and can be selected as the optimised number of clusters.

8.4.1.2 The Silhouette score

In addition to the elbow method, the Davies-Bouldin Index and silhouette index are also plotted in this research to validate this finding.

```
from sklearn.metrics import silhouette_score

silhouette_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(scaled_features)
    silhouette_scores.append(silhouette_score(scaled_features, labels))

# Plot silhouette scores
plt.figure(figsize=(8, 6))
plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.title('Silhouette Score for Optimal Number of Slices')
plt.xlabel('Number of Slices (k)')
plt.ylabel('Silhouette Score')
plt.show()
```

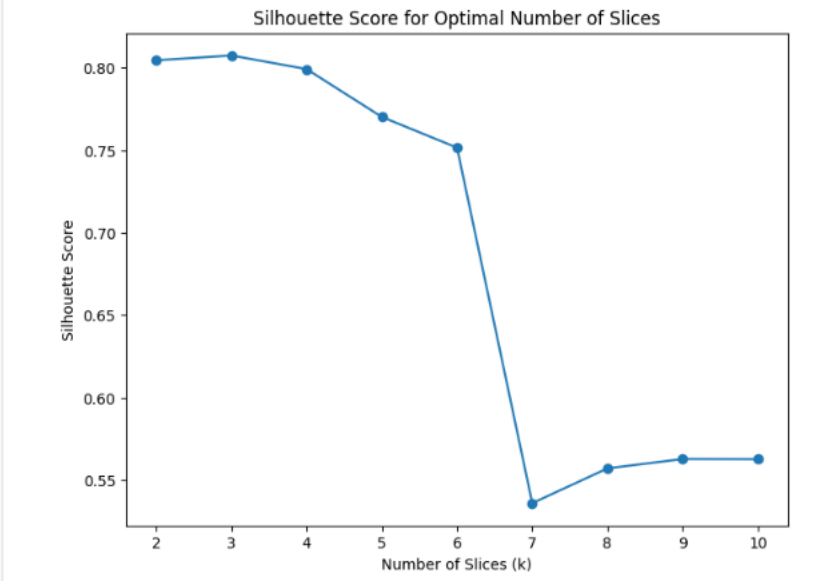


Figure 8.4.2: Silhouette Score to Determine Number of Slices

The Silhouette score ranges between -1 to 1 where number of slices with a value closest to 1 indicates that the slices are well defined and well separated. The number of slices with a silhouette score closer to 0 indicates that they might have overlapping features and are very close to the slice's decision boundary. They are not well defined. A score closer to -1 indicates that the slices might be in the wrong cluster. The features making up the slice might be allocated wrongly. From the chart above, there is no number of clusters below 0 which shows that features making up the slices were correctly allocated. 2, 3, and 4 number of slices have the highest silhouette score with 3 having the highest. There is a significant drop at 6 which indicates these less optimised. Based on this silhouette scores, 3 is identified as the optimised number of slices.

8.4.1.3 The Davies-Bouldin Index

```
from sklearn.metrics import davies_bouldin_score

db_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(scaled_features)
    db_scores.append(davies_bouldin_score(scaled_features, labels))

# Plot Davies-Bouldin index
plt.figure(figsize=(8, 6))
plt.plot(range(2, 11), db_scores, marker='o')
plt.title('Davies-Bouldin Index for Optimal Number of Slices')
plt.xlabel('Number of Slices (k)')
plt.ylabel('Davies-Bouldin Index')
plt.show()
```

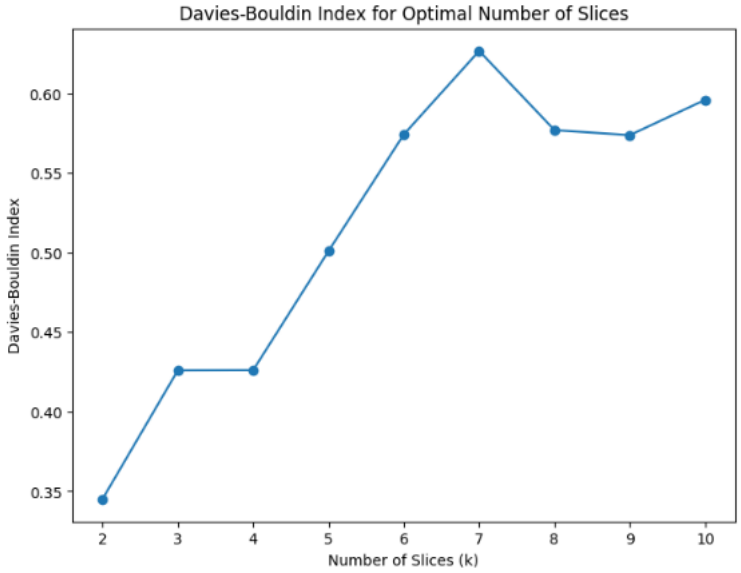


Figure 8.4.3: Davies-Bouldin Index to Determine Number of Slices

With Davies-Bouldin Index values, the lower the value the better the clustering solution. Davies-Bouldin Index values measure the separation between the slice clusters as well as the compactness of the features in the slices. Values closer to 0 indicates well defined compact and well separated clusters. 2 slices have the lowest Davies-Bouldin Index however, 3 also has a relatively low value which indicates that the optimised number of slices could be either 2 or 3.

Table 8.4.1. Results Of Charts for Number Of Slices Selection

Chart	Elbow Method	Silhouette Score	Davies-Bouldin Index
Suggested Cluster	3	3	2, 3

Multiple metrics suggested three as the number of slices and was used in the validation of the model.

8.5 Slice Allocation

The slice allocation module includes the feature of analysing available resource and orchestrating the most suitable alternative slice to meet the user equipment requirement thereby reducing the risk of denial and improving QoE. It also includes the classification of user equipment with their requirements and allocation of slices to meet the network slice requirement of each category or group. These modules are evaluated by analysing the resulting output of the eSS module and understanding if the accuracy of the allocated slice. A clustering machine learning algorithm, K-Means was used to group and formulate the slice based on bandwidth, latency, throughput, jitter, data volume, data rate, and speed. The number of slices was previously determined in Chapter 8.4 and was used as the k value for clustering. The resulting slices were then assigned to each application per row in the dataset populating the slice column.

```
Slice Centers:
      Bandwidth    Latency    Throughput      DataRate        Jitter  \
0  1.902595e+07   2.689860  1.893583e+07  1.522076e+08  3.949895e+05
1  3.421186e+09   0.000002  3.421143e+09  2.736949e+10  3.108647e-01
2  4.034589e+03  58.455930  2.406698e+03  3.227671e+04  1.092745e+07

     Packet_Loss    Data_Volume         Speed  Packet_Delay   Reliability
0  -14399.046879   50023.506957  1.902595e+07  1.035680e-01  1.602255e+05
1 -180819.095238    5069.971429  3.421186e+09  5.576190e-07  7.137949e+06
2  -21402.760152  317262.811494  4.034589e+03  3.185655e+00  8.668327e+04

ProtocolName and Assigned Slice:
         ProtocolName  Slice
10112            HTTP      0
995               TOR      0
14045      OFFICE_365      0
16665    MS_ONE_DRIVE      0
16396      CLOUDFLARE      0
13965           SKYPE      0
12021     SSL_NO_CERT      0
19254       WIKIPEDIA      0
6730     APPLE_ICLOUD      0
6814            YAHOO      0
13037       MICROSOFT      0
8592         FACEBOOK      0
5184          DROPBOX      0
7139            GMAIL      0
9042              MSN      0
16648      HTTP_PROXY      0
7570          YOUTUBE      0
5524     HTTP_CONNECT      0
6098           GOOGLE      0
279    WINDOWS_UPDATE      0
16354          AMAZON      0
2630              SSL      0
2615            MSSQL      0
8739          TWITTER      0
6748    CONTENT_FLASH      0
4863              SSL      1
14538      HTTP_PROXY      1
18039            HTTP      1
18509         TWITTER      2
16254       MICROSOFT      2
18005  WINDOWS_UPDATE      2
9144           AMAZON      2
14649             MSN      2
14505           GMAIL      2
19047            HTTP      2
10727      CLOUDFLARE      2
6824            YAHOO      2
10262         DROPBOX      2
18932        FACEBOOK      2
18676         YOUTUBE      2
13421           SKYPE      2
15347    HTTP_CONNECT      2
19285      HTTP_PROXY      2
10647             SSL      2
10779          GOOGLE      2
```

Figure 8.5.1: Slice Centers and Allocation

Figure 8.5.1 shows the characteristic of each slice, the bandwidth, latency, and other metrics mean value that each slice can supply. The diagram also shows each application, and the slice allocated to

them. For example, Slice 0 was orchestrated for HTTP traffic, Slice 1 was orchestrated for SSL traffic and slice 2 was orchestrated for YouTube traffic. A sequence of network functions is chained for each slice as service chaining to meet the specific requirements of each slice.

8.5.1 Analysis for Each Slice Network Functions Requirements:

Slice 0: Required Network Functions:

- Bandwidth Management: Ensure optimised use of available bandwidth (2.18 Mbps).
- Latency Optimization: Minimize latency to 1.36 ms.
- Throughput Optimization: Maximize throughput to 2.16 Mbps.
- Jitter Control: Stabilize variations in packet arrival times (168,392 ns).
- Reliability Enhancement: Ensure high network reliability (15,306).

Slice 1: Required Network Functions:

- Bandwidth Management: Ensure optimised use of available bandwidth (3.42 Gbps).
- Latency Optimization: Achieve ultra-low latency (2 µs).
- Throughput Optimization: Maximize throughput to 3.42 Gbps.
- Jitter Control: Minimize jitter (0.31 ns).
- Reliability Enhancement: Ensure high network reliability (7,137,949).

Slice 2: Required Network Functions:

- Bandwidth Management: Ensure optimised use of available bandwidth (4.03 Kbps).
- Latency Optimization: Minimize latency to 54.45 ms.
- Throughput Optimization: Maximize throughput to 4.06 Kbps.
- Jitter Control: Stabilize variations in packet arrival times (10,927,450 ns)
- Reliability Enhancement: Ensure high network reliability (86,683).

Service Chaining:

Service Chaining is a mechanism that sequences multiple network functions in a defined order to achieve specific performance goals in a network. In this study, a service chain is designed to enhance network performance by dynamically managing bandwidth, optimizing latency, improving throughput, controlling jitter, and enhancing reliability. The network functions are executed in the following sequence:

1. Bandwidth Management Service Network Function – Regulates and allocates bandwidth to ensure efficient resource utilization based on real-time network demand.
2. Latency Optimization Service Network Function – Reduces delay by optimizing data packet routing and processing, ensuring minimal transmission time.
3. Throughput Optimization Service Network Function – Enhances data transfer efficiency by managing congestion and optimizing data flow.

4. Jitter Control Service Network Function – Mitigates variations in packet arrival times, ensuring stable and predictable network performance.
5. Reliability Enhancement Service Network Function – Improves overall network resilience by implementing fault tolerance mechanisms and redundancy strategies.

By chaining these services in this specific order, the system ensures that network resources are efficiently managed while maintaining low latency, high throughput, minimal jitter, and improved reliability for critical applications.

The total number of applications assigned to each slice for the 9,741 live applications in part1 dataset are shown in Figure 8.5.2.

```
total_rows = len(part1)
print(f'Total number of rows in the dataset: {total_rows}')

# Create a DataFrame for slice 0
cluster_0_data = part1[part1['Best_Fit_Cluster'] == 0]
cluster_0_stats = {
    'Slice': [0],
    'Total_Items': [len(cluster_0_data)],
    'Unique_Items': [cluster_0_data['ProtocolName'].nunique()]
}

# Create a DataFrame for slice 1
cluster_1_data = part1[part1['Best_Fit_Cluster'] == 1]
cluster_1_stats = {
    'Slice': [1],
    'Total_Items': [len(cluster_1_data)],
    'Unique_Items': [cluster_1_data['ProtocolName'].nunique()]
}

# Create a DataFrame for slice 2
cluster_2_data = part1[part1['Best_Fit_Cluster'] == 2]
cluster_2_stats = {
    'Slice': [2],
    'Total_Items': [len(cluster_2_data)],
    'Unique_Items': [cluster_2_data['ProtocolName'].nunique()]
}

# Combine the results into a single DataFrame
result_df_cluster = pd.concat([pd.DataFrame(cluster_0_stats), pd.DataFrame(cluster_1_stats), pd.DataFrame(cluster_2_stats)], ignore_index=True)
# Combine the results into a single DataFrame
print(result_df_cluster)
# Total number of items for each applicaton in each slice
total_items_per_protocol_cluster = part1.groupby(['Best_Fit_Slice', 'ProtocolName']).size().reset_index(name='Total_Items')
# Number of unique items for each applcation in each slice
unique_items_per_protocol_cluster = part1.groupby(['Best_Fit_Slice', 'ProtocolName'])['Best_Fit_Cluster'].nunique().reset_index(name='Unique_Items')
# Combine the results into a DataFrame for better readability
result_df_cluster = pd.merge(total_items_per_protocol_cluster, unique_items_per_protocol_cluster, on=['Best_Fit_Slice', 'ProtocolName'])
print(result_df_cluster)
```

```
Total number of rows in the dataset: 9741
   Slice  Total_Items  Unique_Items
0      0         9271            25
1      1           35             3
2      2          435            17
    Best_Fit_Slice    ProtocolName  Total_Items  Unique_Items
0                0          AMAZON          124             1
1                0   APPLE_ICLOUD            2             1
2                0     CLOUDFLARE            6             1
3                0  CONTENT_FLASH           74             1
4                0        DROPBOX           13             1
5                0       FACEBOOK           66             1
6                0           GMAIL           45             1
7                0         GOOGLE         1404             1
```

Figure 8.5.2: Applications And Slice Distribution In Dataset

```
[77] unique_protocols_per_cluster = part1.groupby('Slice')['ProtocolName'].unique().tolist()
     for slice, protocols in enumerate(unique_protocols_per_cluster):
         print(f"Applications in Slice {slice}:")
         print(protocols)
```

```
Applications in Slice 0:
['HTTP' 'YOUTUBE' 'HTTP_CONNECT' 'GOOGLE' 'HTTP_PROXY' 'AMAZON' 'SSL'
 'CONTENT_FLASH' 'TWITTER' 'WINDOWS_UPDATE' 'GMAIL' 'DROPBOX' 'FACEBOOK'
 'MICROSOFT' 'YAHOO' 'MSN' 'WIKIPEDIA' 'SSL_NO_CERT' 'SKYPE' 'CLOUDFLARE'
 'MS_ONE_DRIVE' 'OFFICE_365' 'TOR' 'APPLE_ICLOUD' 'MSSQL']
Applications in Slice 1:
['HTTP_PROXY' 'HTTP' 'SSL']
Applications in Slice 2:
['MSN' 'HTTP_PROXY' 'HTTP_CONNECT' 'HTTP' 'MICROSOFT' 'GOOGLE' 'SSL'
 'TWITTER' 'SKYPE' 'YOUTUBE' 'FACEBOOK' 'DROPBOX' 'YAHOO' 'CLOUDFLARE'
 'GMAIL' 'AMAZON' 'WINDOWS_UPDATE']
```

Figure 8.5.3: Applications Per Slice

Figure 8.5.3 shows the applications assigned to each slice. There may be some traffic data with different slices, even though the protocols are the same, but their network requirement might be different. For example, downloading a file via FTP might be downloading a file as small as 5mb or as large as 5Gb although they are both using same protocol hence a protocol name or application might appear in different slices based on their requirement. They are identified with their indexes. Figure 8.5.4 shows applications on the network with the evaluated applications' network function requirement for an efficiently orchestrated 5G slice.

```
+ Code + Text

columns_for_clustering = ['Bandwidth', 'Latency', 'Throughput', 'DataRate', 'Jitter', 'Packet_Loss', 'Data_Volume', 'Speed', 'Pac

# Define a function to categorize values into low, medium, and high
def categorize(value, low_threshold, high_threshold):
    if value < low_threshold:
        return 'Low'
    elif low_threshold <= value <= high_threshold:
        return 'Medium'
    else:
        return 'High'

# Categorize cluster centers into low, medium, and high
for col in columns_for_clustering:
    low_threshold = result[col].quantile(0.25)
    high_threshold = result[col].quantile(0.75)
    result[col] = result[col].apply(lambda x: categorize(x, low_threshold, high_threshold))

# Display the categorized slice features
print("Categorized Applications Slice Requirements:")
print(result)
```

```
Categorized Applications Slice Requirements:
      ProtocolName Bandwidth Latency Throughput DataRate  Jitter Packet_Loss  \
0           AMAZON    Medium  Medium     Medium   Medium  Medium         Low
1    CONTENT_FLASH       Low     Low        Low      Low     Low      Medium
2          DROPBOX      High     Low       High     High     Low        High
3         FACEBOOK    Medium  Medium     Medium   Medium  Medium        High
4            GMAIL    Medium  Medium     Medium   Medium  Medium      Medium
5           GOOGLE    Medium  Medium     Medium   Medium    High      Medium
6             HTTP      High  Medium       High     High  Medium      Medium
7     HTTP_CONNECT    Medium  Medium     Medium   Medium  Medium      Medium
8       HTTP_PROXY      High  Medium       High     High  Medium      Medium
9        MICROSOFT    Medium  Medium     Medium   Medium  Medium         Low
10             MSN       Low  Medium        Low      Low    High         Low
11           MSSQL    Medium    High     Medium   Medium  Medium         Low
12    MS_ONE_DRIVE    Medium    High     Medium   Medium  Medium         Low
13           SKYPE       Low    High        Low      Low    High      Medium
14             SSL      High  Medium       High     High  Medium      Medium
15     SSL_NO_CERT    Medium     Low     Medium   Medium     Low        High
16             TOR      High     Low       High     High     Low        High
17         TWITTER       Low  Medium        Low      Low  Medium      Medium
18  WINDOWS_UPDATE    Medium     Low     Medium   Medium     Low        High
19           YAHOO       Low    High        Low      Low    High      Medium
20         YOUTUBE    Medium    High     Medium   Medium    High      Medium

   Data_Volume   Speed Packet_Delay Reliability
0       Medium  Medium       Medium        High
1          Low     Low       Medium      Medium
2          Low    High          Low      Medium
3          Low  Medium         High         Low
4         High  Medium       Medium      Medium
5       Medium  Medium         High      Medium
6       Medium    High       Medium        High
7       Medium  Medium       Medium      Medium
8         High    High       Medium        High
9       Medium  Medium       Medium      Medium
```

Figure 8.5.4: Network Function Distribution by Monitored Application

8.5.2 The Individual Metric Importance and Result

Further investigation into the importance of each metric in orchestrating the slice based on the evaluation of the applications on the monitored network and their data request validates the optimised and efficient utilisation of network resource to improve the slice QoS. Figure 8.5.5 shows the metric importance and impact on the decision of orchestrating the slice which was obtained from the machine learning RandomForestClassifier algorithm. Jitter was seen as the metric with the highest importance which contributed to the decision making in allocating slices to applications. This shows that the model put importance on the time delay between when the application requested a slice and when the appropriate slice is delivered to improve user experience. The variance in latency was of upmost importance and was highly considered. The result also shows Latency as the next important metric considered by the model in predicting the slice allocation. Packet delay was the third important

metric, and these top three metrics are related which validates the behaviour of the model. The fourth important feature considered by the metric was throughput which is ensuring the amount of data required to be transported is moved successfully within a given period. This improves user experience and the slice QoS. The fifth and sixth important features considered as shown by the results are Bandwidth and DataRate. These are followed by the other metrics.

```
# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Get feature importances
feature_importances = pd.DataFrame(rf_classifier.feature_importances_, index=features.columns, columns=['Importance'])
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

# Display feature importances
print(feature_importances)
```

```
              Importance
Jitter          0.377905
Latency         0.248795
Packet_Delay    0.190662
Throughput      0.046817
Bandwidth       0.036923
DataRate        0.032923
Speed           0.027815
Reliability     0.018018
Data_Volume     0.012272
Packet_Loss     0.007869
```

Figure 8.5.5: Metrics Decision Importance

8.5.3 Most Suitable Alternative Slice Analysis

In the event the available resources are not sufficient for the best fit slice, the proposed AI model pipeline will orchestrate an alternative slice to ensure smooth user experience by utilising lesser network resource yet fulfilling the need of the application on the network. Figure 8.5.6 shows a diagram of the applications with the best slice allocation and an alternative slice allocation to effectively manage available network resources, improve QoS and reduce the risk of denial of slice.

```
from sklearn.metrics.pairwise import euclidean_distances

# Calculate distances to all cluster centers
distances = euclidean_distances(features_scaled, kmeans.cluster_centers_)

# Find the second minimum distance and corresponding cluster for each Applcation
alternative_slices = []
for i, row in enumerate(distances):
    best_fit_slice = best_fit_slices[i]
    second_min_distance_cluster = np.argpartition(row, 1)[1]
    alternative_slices.append(second_min_distance_cluster if second_min_distance_cluster != best_fit_slice else np.argpartition(row, 2)[2])

# Add the 'Alternative_Slice' column to the DataFrame
part1['Alternative_Slice'] = alternative_slices

# Display the data with assigned clusters and alternative clusters
print("Application with Best Fit and Alternative Slices:")
print(part1[['ProtocolName', 'Best_Fit_Slice', 'Alternative_Slice']])
```

```
Application with Best Fit and Alternative Slices:
         ProtocolName  Best_Fit_Slice  Alternative_Slice
10112            HTTP               0                  2
11017            HTTP               0                  2
7570          YOUTUBE               0                  2
5524     HTTP_CONNECT               0                  2
11105    HTTP_CONNECT               0                  2
4565             HTTP               0                  2
296           YOUTUBE               0                  2
5405             HTTP               0                  2
2357             HTTP               0                  2
16206    HTTP_CONNECT               0                  2
10857    HTTP_CONNECT               0                  2
7013          YOUTUBE               0                  2
14649             MSN               2                  0
6098           GOOGLE               0                  2
11224    HTTP_CONNECT               0                  2
12243            HTTP               0                  2
16648      HTTP_PROXY               0                  2
17195          GOOGLE               0                  2
333              HTTP               0                  2
14776    HTTP_CONNECT               0                  2
8908          YOUTUBE               0                  2
5180             HTTP               0                  2
2943           GOOGLE               0                  2
17705            HTTP               0                  2
16354          AMAZON               0                  2
8135       HTTP_PROXY               0                  2
1234       HTTP_PROXY               0                  2
16721      HTTP_PROXY               0                  2
3561           GOOGLE               0                  2
16507            HTTP               0                  2
5169             HTTP               0                  2
5991             HTTP               0                  2
19285      HTTP_PROXY               2                  0
3867             HTTP               0                  2
```

Figure 8.5.6: Alternative Network Slice

The model can orchestrate one of the standard 3GPP 5G network slices if the application on the network requires any of them. The uRLLC, eMBB and mMTC characteristics are programmed in the model pipeline. If the required network slice features match any of the standard 5G network slice, then the network templates for the slice is defined and orchestrated, however if a bespoke slice is required or if the available network resource available is limited, then a bespoke slice is used to form the slice template and orchestrated. Figure 8.5.7 shows the output of the model with the slice column showing if the slice is a bespoke slice or one of the standard 3GPP 5G network slice. The output shown in Figure 8.5.5 is from the part1 dataset used in evaluating the model.

```python
# Define characteristics for each slice
slice_characteristics = {
    'uRLLC': {'Latency': 'Low', 'Throughput': 'High', 'Reliability': 'High'},
    'eMBB': {'Throughput': 'High', 'Latency': 'Medium', 'Reliability': 'Medium'},
    'mMTC': {'Throughput': 'Medium', 'Latency': 'Medium', 'Reliability': 'Low'},
}

# Function to categorize slice
def categorize_cluster(row):
    for slice_name, characteristics in slice_characteristics.items():
        match = all(row[feature] == value for feature, value in characteristics.items())
        if match:
            return slice_name
    return 'Bespoke'

# Add a new column for the categorized slice
categorized_centers['Slice'] = categorized_centers.apply(categorize_cluster, axis=1)

# Display the categorized slices
print(categorized_centers[[ 'Slice']])
print(categorized_centers)
```

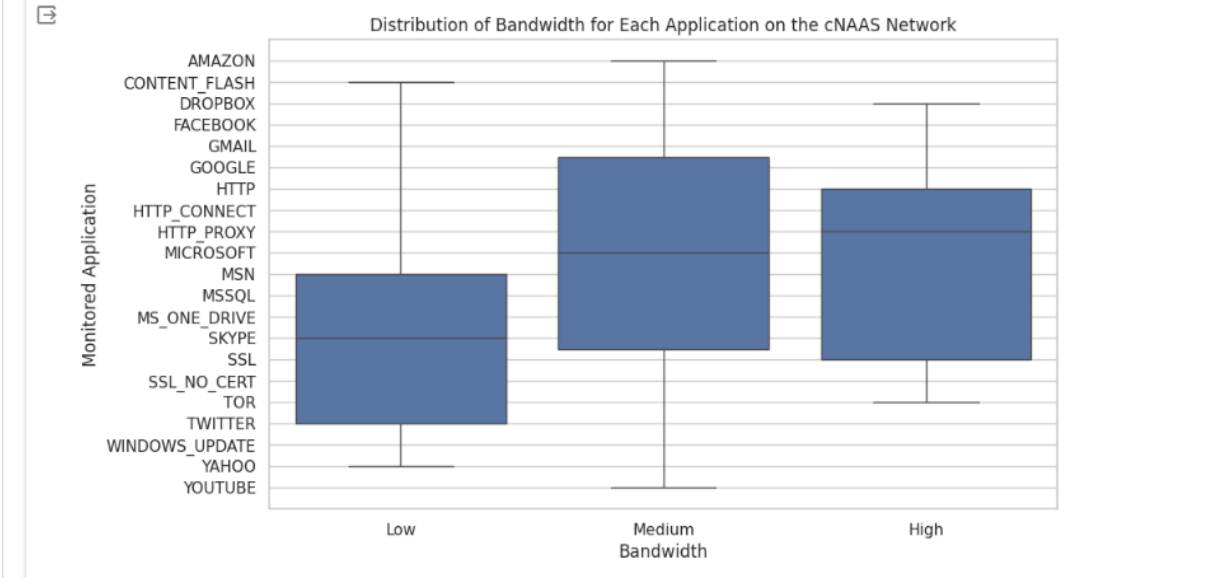
```
     Slice
0  Bespoke
1    uRLLC
2  Bespoke
  Bandwidth Latency Throughput DataRate  Jitter Packet_Loss Data_Volume  \
0    Medium  Medium     Medium   Medium  Medium        High      Medium
1      High     Low       High     High     Low         Low         Low
2       Low    High        Low      Low    High      Medium        High

    Speed Packet_Delay Reliability    Slice
0  Medium       Medium      Medium  Bespoke
1    High          Low        High    uRLLC
2     Low         High         Low  Bespoke
```

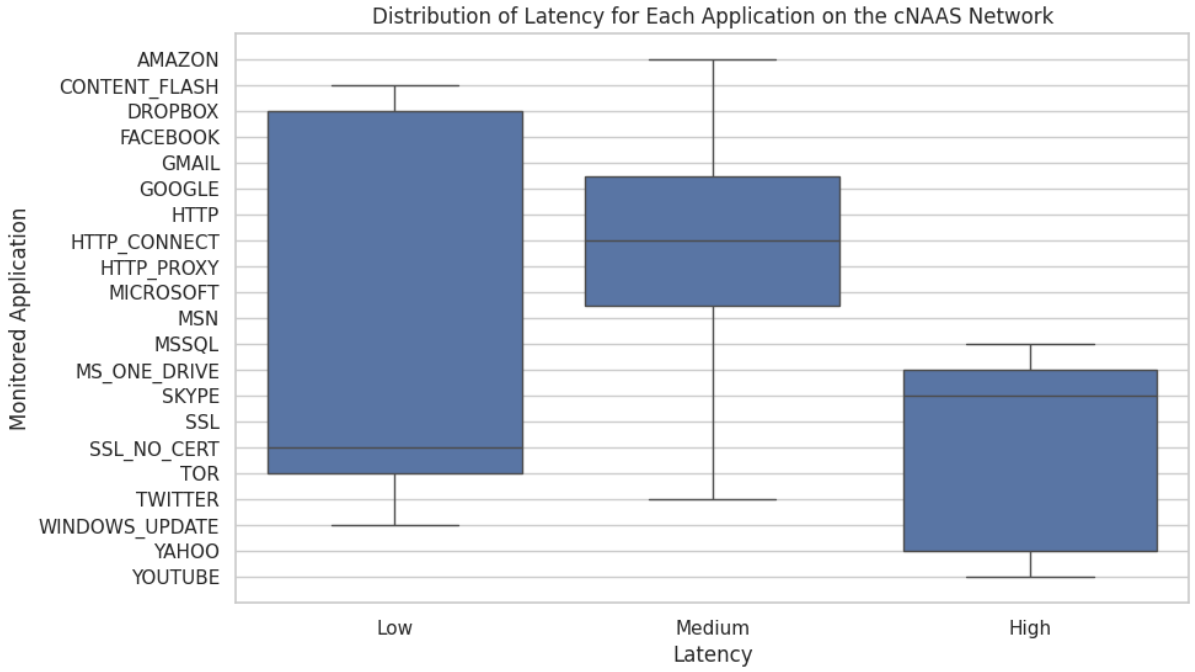
Figure 8.5.7: Standard and Bespoke 5G Slice

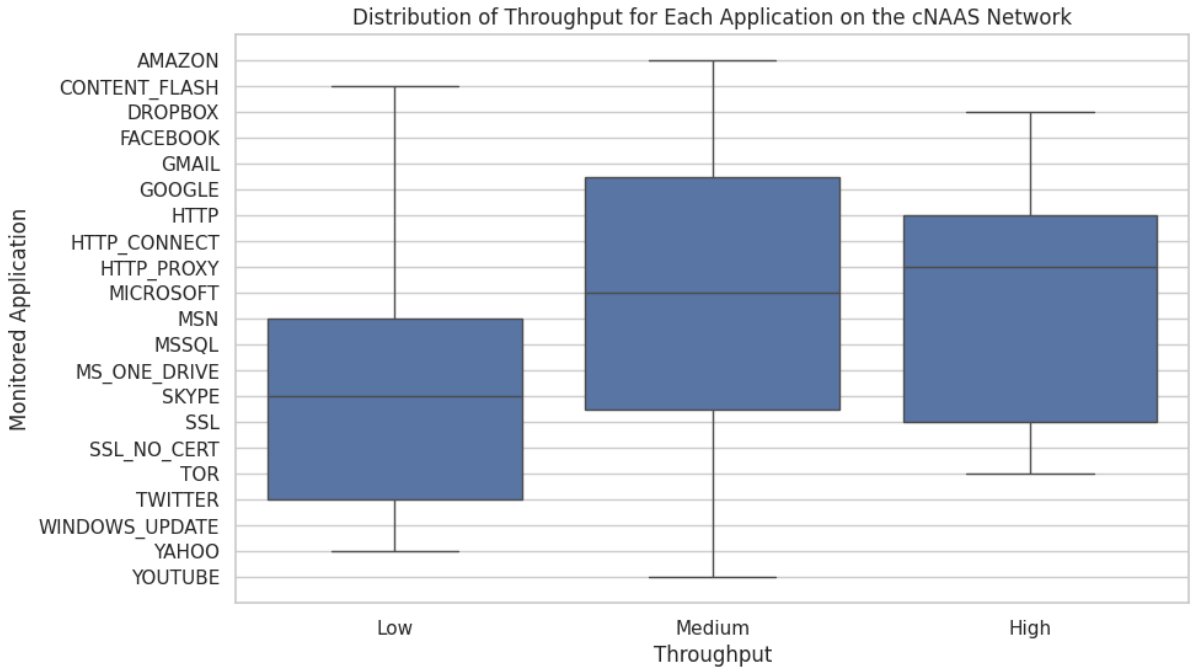
Figure 8.5.7 shows that slice 1 which was indicated as the most expensive from the previous analysis is a standard 5G network slice called uRLLC. The other two slices orchestrated are bespoke slices with custom network function configurations.

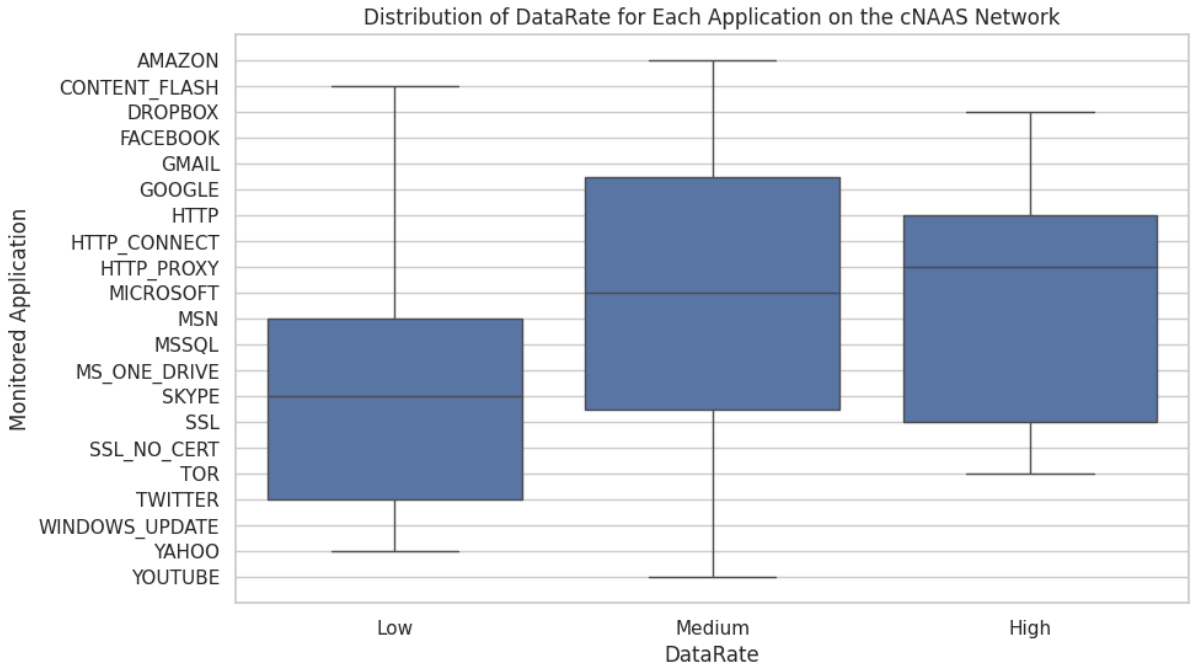
Figure 8.5.8 shows the distribution of each network slice feature using boxplots. Applications on the network are grouped by the slices allocated to them. The boxplots provide insight into how these slice features are distributed across the diverse identified applications on the network.

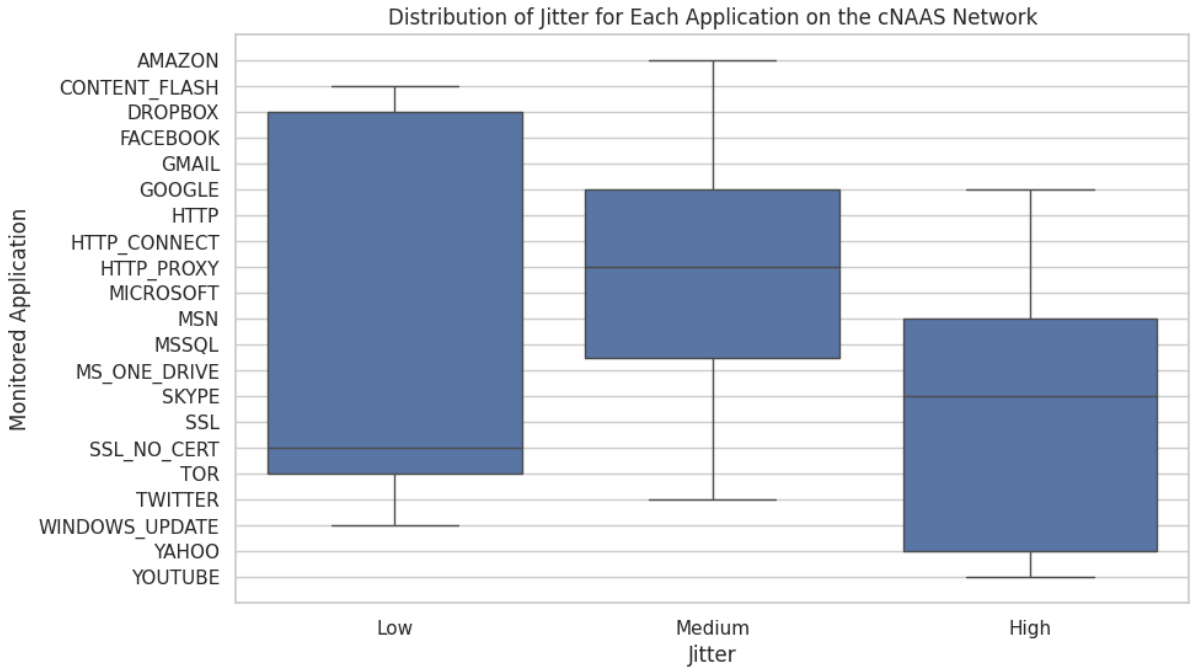
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Set the style for the plots
sns.set(style="whitegrid")

# Iterate through each category and create box plots
categories = ['Bandwidth', 'Latency', 'Throughput', 'DataRate', 'Jitter', 'Packet_Loss', 'Data_Volume', 'Speed', 'Packet_Delay', 'Reliability']

for category in categories:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=category, y='ProtocolName', data=result, order=['Low', 'Medium', 'High'])
    plt.title(f'Distribution of {category} for Each Application on the cNAAS Network')
    plt.ylabel('Monitored Application')
    plt.show()
```

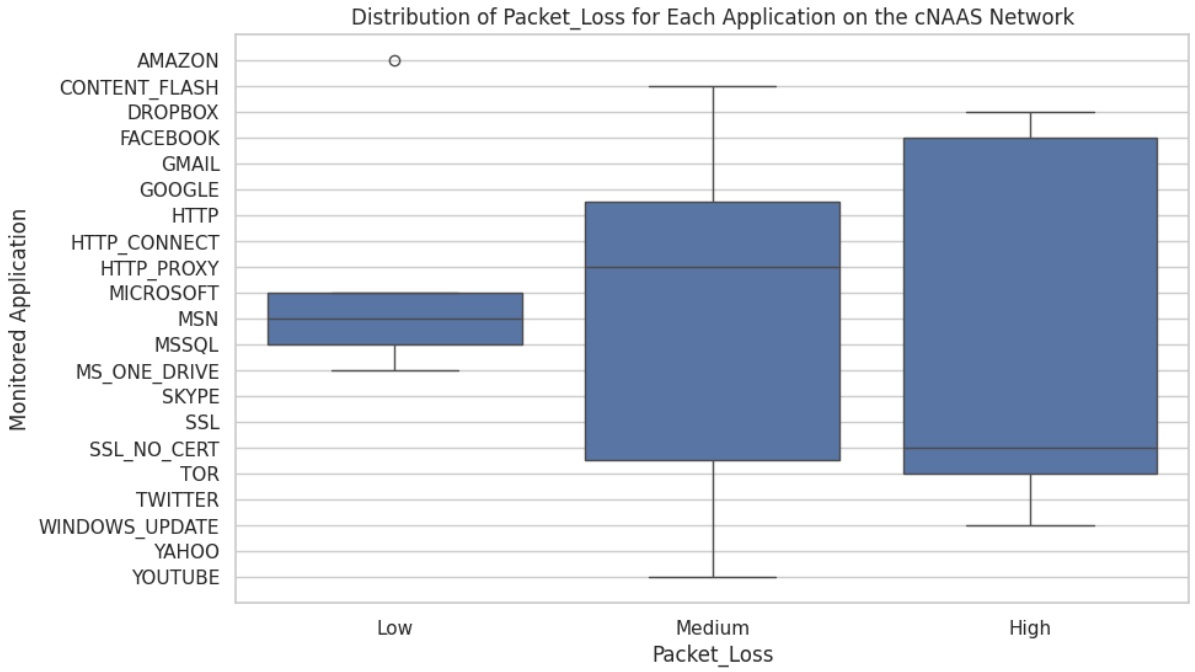


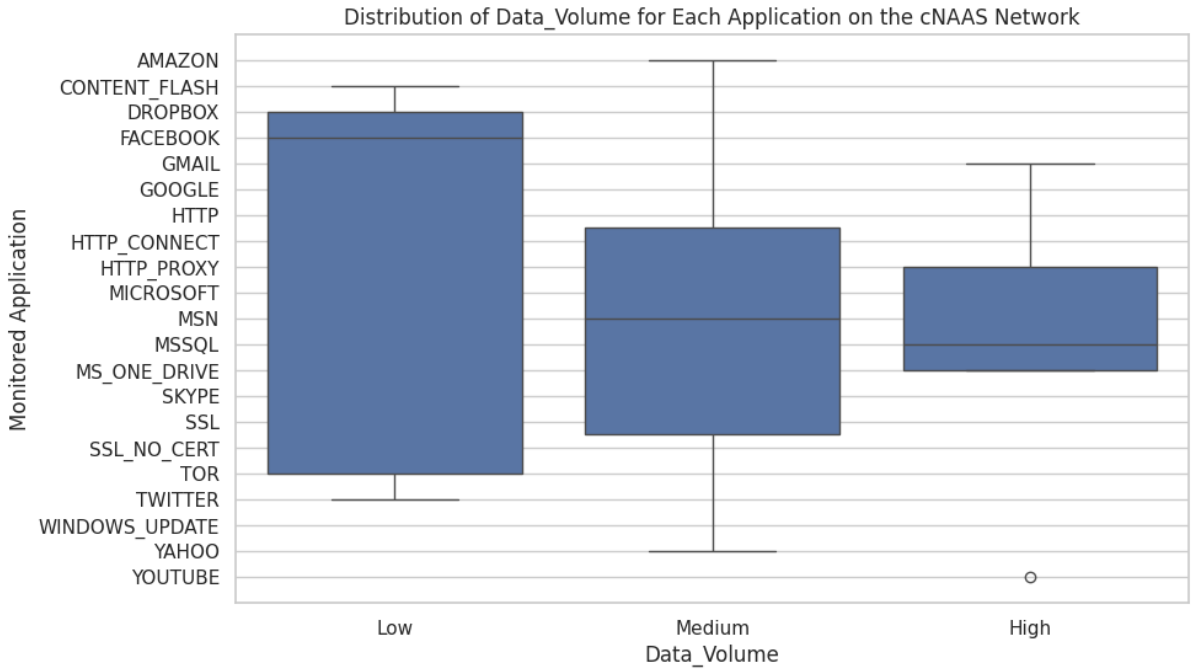


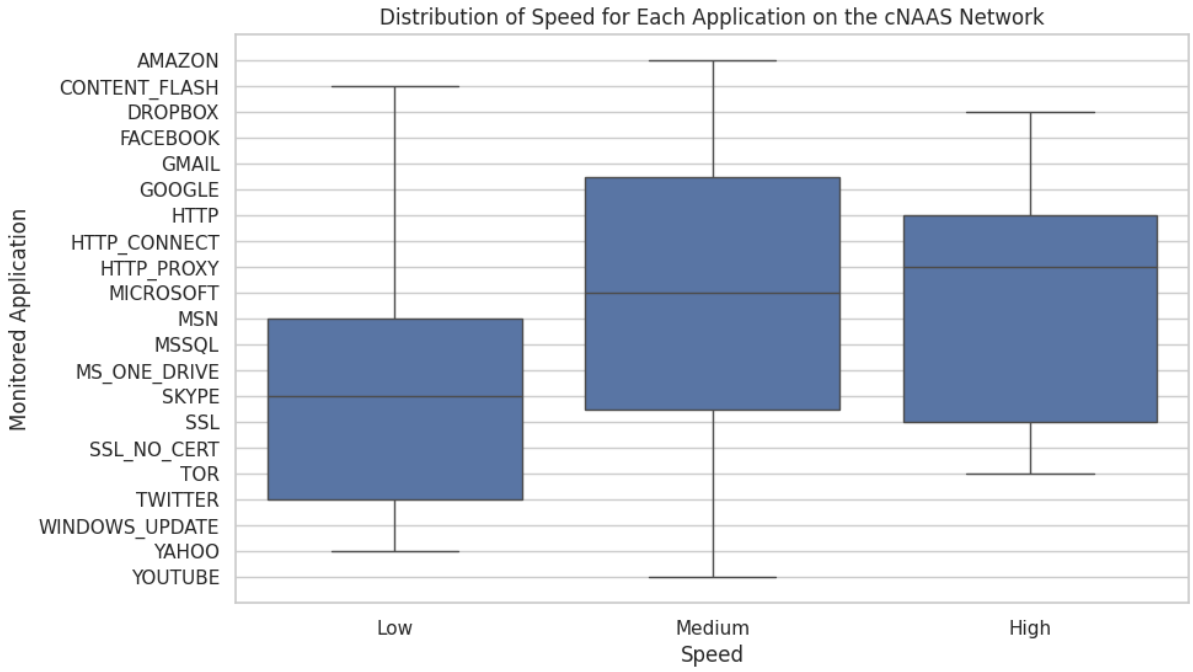

Distribution of Throughput for Each Application on the cNAAS Network
Monitored Application
AMAZON
CONTENT_FLASH
DROPBOX
FACEBOOK
GMAIL
GOOGLE
HTTP
HTTP_CONNECT
HTTP_PROXY
MICROSOFT
MSN
MSSQL
MS_ONE_DRIVE
SKYPE
SSL
SSL_NO_CERT
TOR
TWITTER
WINDOWS_UPDATE
YAHOO
YOUTUBE
Low
Medium
High
Throughput


Distribution of DataRate for Each Application on the cNAAS Network
Monitored Application
AMAZON
CONTENT_FLASH
DROPBOX
FACEBOOK
GMAIL
GOOGLE
HTTP
HTTP_CONNECT
HTTP_PROXY
MICROSOFT
MSN
MSSQL
MS_ONE_DRIVE
SKYPE
SSL
SSL_NO_CERT
TOR
TWITTER
WINDOWS_UPDATE
YAHOO
YOUTUBE
Low
Medium
High
DataRate


Distribution of Jitter for Each Application on the cNAAS Network
Monitored Application
AMAZON
CONTENT_FLASH
DROPBOX
FACEBOOK
GMAIL
GOOGLE
HTTP
HTTP_CONNECT
HTTP_PROXY
MICROSOFT
MSN
MSSQL
MS_ONE_DRIVE
SKYPE
SSL
SSL_NO_CERT
TOR
TWITTER
WINDOWS_UPDATE
YAHOO
YOUTUBE
Low
Medium
High
Jitter


Distribution of Packet_Loss for Each Application on the cNAAS Network
Monitored Application
AMAZON
CONTENT_FLASH
DROPBOX
FACEBOOK
GMAIL
GOOGLE
HTTP
HTTP_CONNECT
HTTP_PROXY
MICROSOFT
MSN
MSSQL
MS_ONE_DRIVE
SKYPE
SSL
SSL_NO_CERT
TOR
TWITTER
WINDOWS_UPDATE
YAHOO
YOUTUBE
Low
Medium
High
Packet_Loss


Distribution of Data_Volume for Each Application on the cNAAS Network
Monitored Application
AMAZON
CONTENT_FLASH
DROPBOX
FACEBOOK
GMAIL
GOOGLE
HTTP
HTTP_CONNECT
HTTP_PROXY
MICROSOFT
MSN
MSSQL
MS_ONE_DRIVE
SKYPE
SSL
SSL_NO_CERT
TOR
TWITTER
WINDOWS_UPDATE
YAHOO
YOUTUBE
Low
Medium
High
Data_Volume


Distribution of Speed for Each Application on the cNAAS Network
Monitored Application
AMAZON
CONTENT_FLASH
DROPBOX
FACEBOOK
GMAIL
GOOGLE
HTTP
HTTP_CONNECT
HTTP_PROXY
MICROSOFT
MSN
MSSQL
MS_ONE_DRIVE
SKYPE
SSL
SSL_NO_CERT
TOR
TWITTER
WINDOWS_UPDATE
YAHOO
YOUTUBE
Low
Medium
High
Speed

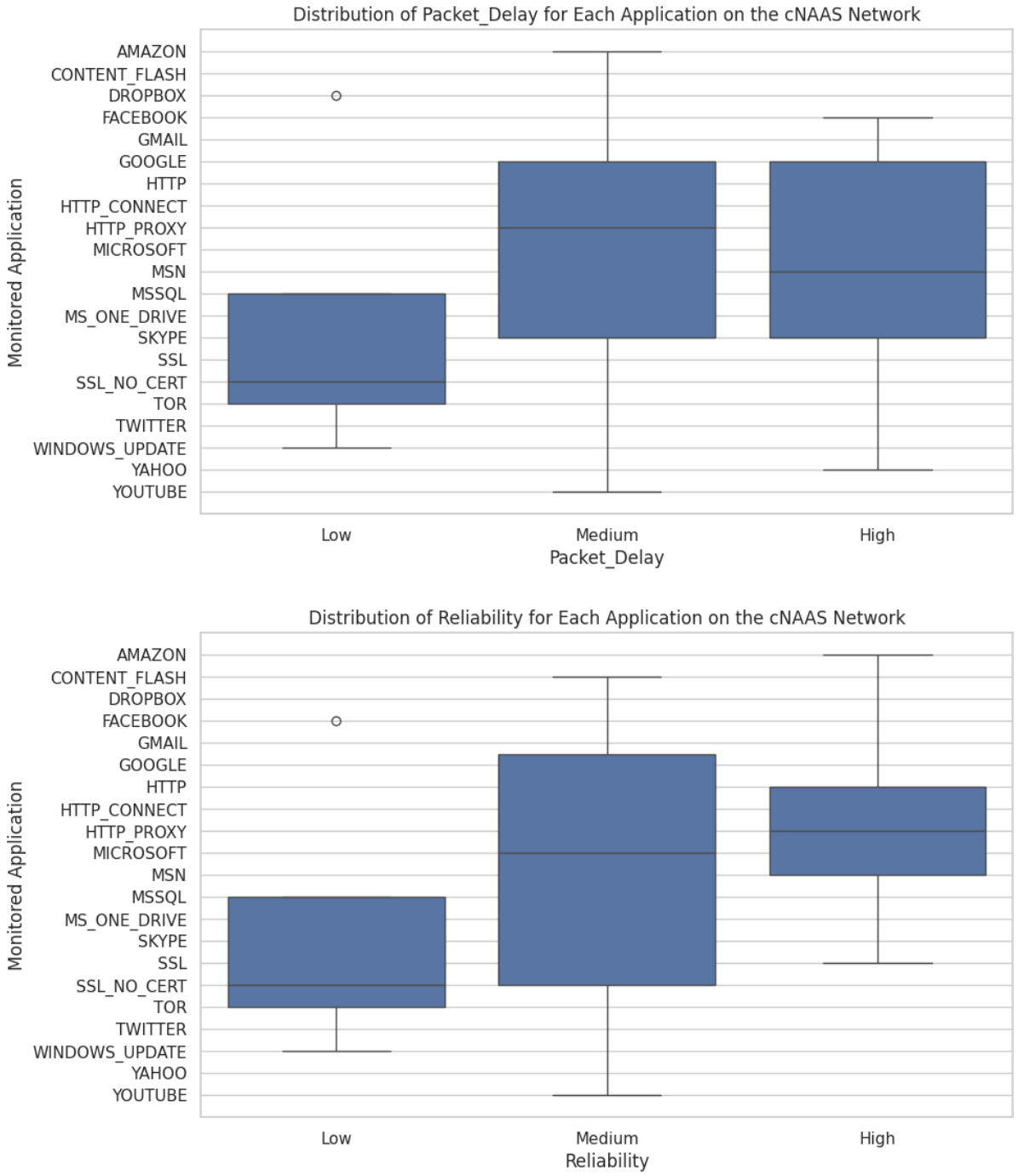


Figure 8.5.8: Distribution of Network Slice Features Visualisation With Boxplot

Table 8.5.1 shows the breakdown of the standard 5G network slice features.

Table 8.5.1. Characteristics of 5G Standard Slice

5G Slice	Latency	Throughput	Reliability	Jitter	Bandwidth	DataRate
uRLLC	Low	High	High	Low	High	High
eMBB	Medium	High	Medium	Medium	Medium	Medium

mMTC	Medium	Medium	Low	High	Low	Low

Table 8.5.2. Low, Medium, High Categorisation of Network Slices

Metric	Bandwidth	Latency	Throughput	DataRate	Jitter	Packet Loss	Data Volume	Speed	Packet Delay	Reliability
Slice 0	Medium	Medium	Medium	Medium	Medium	High	Medium	Medium	Medium	Medium
Slice 1	High	Low	High	High	Low	Low	Low	High	Low	High
Slice 2	Low	High	Low	Low	High	Medium	High	Low	High	Low

8.5.2 Result

From Figure 8.3.2 and Table 8.5.2, it can be deduced that while the packet loss metric is high for slice 0 all the other metrics such as bandwidth, latency, throughput and other measured metrics are medium. Slice 2 however has low bandwidth which will be cheaper than a medium bandwidth as provided by slice 0. Slice 2 has high latency which will be more expensive than a slice providing lower latency such as slice providing medium latency. Slice 2 has low throughput which will be more cost effective than a slice providing a higher throughput such as slice 0 providing medium throughput. A slice with higher jitter will be more cost effective and requires lesser network resource than a slice with lower jitter. Slice 2 has high jitter while slice 0 has medium jitter hence slice 2 is likely more cost effective than slice 0. A slice with higher packet loss will require lesser network resource than a slice with lower packet loss hence slice 0 with high packet loss will be more cost effective for packet loss network resource than slice 2 with medium packet loss. A slice requiring lesser data volume will require lesser network resource than a slice that requires large data volume hence slice 0 will be more cost effective than slice 2 for data volume network resource. A slice with lower speed packet will require lesser resource than a slice with high speed hence slice 2 will be more cost effective than slice 0 for speed network function resources. A slice with high packet delay will be more cost effective than a slice with low packet delay hence slices 2 with high packet delay will be more cost effective than slice 0 and require lesser network resource. A slice with low reliability will require less network resource than a slice with high reliability hence slice 2 will require less network resource than slice 0 with medium reliability for reliability network resource.

Based on the network slice distribution as shown in Table 8.5.2, Slice 1 will be the most expensive, while Slice 2 will be the least expensive and slice 0 will have an average cost between the two slices. Slice 1 has high requirements across various network characteristics, which indicates that the applications requiring this slice are in need of robust and high performing network resources and network functions. Slice 2 suggests being the least expensive as it has low network requirements across various network characteristics which indicates that the applications requiring this slice are delay sensitive applications. Slice 0 will fall in between the cost of slice 1 and slice 2 with a moderate cost.

- Slice 1 – Most expensive
- Slice 2 - Least expensive
- Slice 0 – Moderate cost

This validates the alternative suggested by the proposed model. Where slice 0 was suggested as best fit, it majorly proposed slice 2 as an alternative slice. Slice 2 requires lesser network resource and functions than slice 0 and still be able to meet the network slice requirements of the applications.

Some applications with slice 1 as best fit were also suggested to function well with slice 0 as alternative in the model using the machine learning model pipeline as shown in Figure 8.5.3.

8.6 LazyPredict Evaluation

The automatic selection of machine learning classification algorithm that best fit the inputted data was shown as Random Forest by the LazyPredict algorithm used in this research. This Chapter will validate this result by using any three other algorithms which had lower accuracy results and comparing their accuracies and output against the selected algorithm by LazyPredict module. Random Forest Classifier will be evaluated and compared against the accuracy of Naïve Bayes, Linear Discriminant Analysis and Logistic Regression. Figure 8.6.1.1 shows the accuracy of Naïve Bayes as 0.27. In Figure 8.6.1.2, Linear Discriminant Analysis shows accuracy as 0.996. Random Forest Classifier produced an accuracy of 0.998 as shown in Figure 8.6.1.3 and Logistic Regression produced an accuracy 0f 0.811 as shown in Figure 8.4.1.4.

8.6.1 Implementing Naïve Bayes

Naïve Bayes was used in the eSS model with the dataset as the default supervised machine learning algorithm. Its low accuracy of 0.27 is shown in Figure 8.6.1.1.

```
[15] from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import accuracy_score
     from sklearn.model_selection import train_test_split

     # Assuming 'imputer' is your SimpleImputer instance and 'features' is your feature matrix

     # Impute missing values in the features for training
     X_imputed = imputer.transform(features)

     # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X_imputed, part1['Slice'], test_size=0.2, random_state=42)

     # Create a Naive Bayes Classifier (Gaussian Naive Bayes for continuous features)
     nb_classifier = GaussianNB()

     # Train the classifier
     nb_classifier.fit(X_train, y_train)

     # Predict
     y_pred = nb_classifier.predict(X_test)

     # Evaluate
     accuracy = accuracy_score(y_test, y_pred)
     print(f'Accuracy: {accuracy}')

     Accuracy: 0.27722772277227725
```

Figure 8.6.1.1: Naïve Bayes

8.6.2 Implementing Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) was evaluated by using it as the default on the eSS model compared to the selected Random Forest and the high accuracy of the Linear Discriminant Analysis as 0.996 is shown in Figure 8.6.1.2. The LDA accuracy is good and high, however, as shown later in the result section, the selected Random Forest algorithm has a higher accuracy.

```
[16] from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
     from sklearn.metrics import accuracy_score
     from sklearn.model_selection import train_test_split

     # Assuming 'imputer' is your SimpleImputer instance and 'features' is your feature matrix

     # Impute missing values in the features for training
     X_imputed = imputer.transform(features)

     # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X_imputed, part1['Slice'], test_size=0.2, random_state=42)

     # Create a Linear Discriminant Analysis (LDA) Classifier
     lda_classifier = LinearDiscriminantAnalysis()

     # Train the classifier
     lda_classifier.fit(X_train, y_train)

     # Predict
     y_pred = lda_classifier.predict(X_test)

     # Evaluate
     accuracy = accuracy_score(y_test, y_pred)
     print(f'Accuracy: {accuracy}')

     Accuracy: 0.9966996699669967
```

Figure 8.6.1.2: Linear Discriminant Analysis

8.6.3 Implementing Logistic Regression

Logistic Regression was the third algorithm used for comparison and evaluation of the default supervised machine learning algorithm selected by the eSS model on the given dataset. Logistic Regression shows an accuracy of 0.611 as in Figure 8.6.1.3.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer  # Make sure to import SimpleImputer if it's not already imported

# Assuming 'imputer' is your SimpleImputer instance and 'features' is your feature matrix

# Impute missing values in the features for training
X_imputed = imputer.transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, part1['Slice'], test_size=0.2, random_state=42)

# Create a Logistic Regression Classifier
lr_classifier = LogisticRegression(random_state=42, max_iter=1000)

# Train the classifier
lr_classifier.fit(X_train, y_train)

# Predict
y_pred = lr_classifier.predict(X_test)

# Evaluate
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

Accuracy: 0.8112211221122112
```

Figure 8.6.1.43: Logistic Regression

8.6.4 Implementing Random Forest Classifier

The selected supervised machine learning algorithm by the eSS model is fed into the eSS model and It is high accuracy is shown in Figure 8.6.1.4 as 0.998.

```
[17] from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score


     X_imputed = imputer.transform(features)

     # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X_imputed, part1['Slice'], test_size=0.2, random_state=42)

     # Create a Random Forest Classifier
     rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

     # Train the classifier
     rf_classifier.fit(X_train, y_train)

     # Predict the clusters for the test set
     y_pred = rf_classifier.predict(X_test)

     # Evaluate the accuracy of the classifier
     accuracy = accuracy_score(y_test, y_pred)
     print(f'Accuracy: {accuracy}')

     Accuracy: 0.998019801980198
```

Figure 8.61.4: Random Forest Classifier

8.6.5 Result

A summary of these results are tabulated in Table 8.6.1.1 with Random Forest Classifier of highest accuracy score of 0.998 thereby validating the result of the LazyPredict model. The table also shows the mean squared error (MSE) of the proposed model's prediction on the testing dataset by indicating how close the application cluster predictions are to the actual value. RandomForestClassifier is seen with the lowest value validating the LazyPredict evaluation as the best fit classifier for the network data. The MSE value is used in the Network Data Analytics Function (NWDAF) to provide the analytics for the monitored network performance and load level. These are used by other network functions such as Network Slice Selection Function and Policy Control function to make informed decisions in efficiently orchestrating the 5G network slice.

Table 8.6.1.1. Summary LazyPredict Validation

ML Classifier	Naïve Bayes	Linear Discriminant Analysis	Random Forest Classifier	Logistic Regression
Accuracy Score	0.27	0.996	0.998	0.81
NWDAF (Mean Squared Error)	41.13	1.71	0.22	6.23

8.7 Varying Dataset Input in eSS Model

8.7.1 Part1 Dataset

The validation of the model was evaluated with three different sets of datasets to evaluate its behaviour when subjected to a different input. The first dataset used for evaluation and validation of the proposed model is labelled part1. Figure 8.7.1.1 shows the accuracy of the model as 0.998 with part1 dataset.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


X_imputed = imputer.transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, part1['Slice'], test_size=0.2, random_state=42)

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Predict the clusters for the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9984607491021037
```

Figure 8.7.1.1 Random Forest Classifier

Figure 8.7.1.2 shows the assigned slices with the colour blue, green and orange. We can see that it shows the slice 0 has low bandwidth with low to medium latency and low to medium throughput. Slice 1 has medium to high bandwidth, low latency, and high throughput. Slice 2 has low bandwidth, high latency and low throughput.

```
from mpl_toolkits.mplot3d import Axes3D

# Assuming 'data' DataFrame has columns 'Bandwidth', 'Latency', 'Throughput', 'DataRate', 'Jitter', and 'Cluster'
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot for each cluster
for cluster in range(num_clusters):
    cluster_data = part1[part1['Slice'] == cluster]
    ax.scatter(cluster_data['Bandwidth'], cluster_data['Latency'], cluster_data['Throughput'], label=f'Slice {cluster}')

ax.set_xlabel('Bandwidth')
ax.set_ylabel('Latency')
ax.set_zlabel('Throughput')
ax.set_title('3D Scatter Plot of Slices')
ax.legend()
plt.show()
```

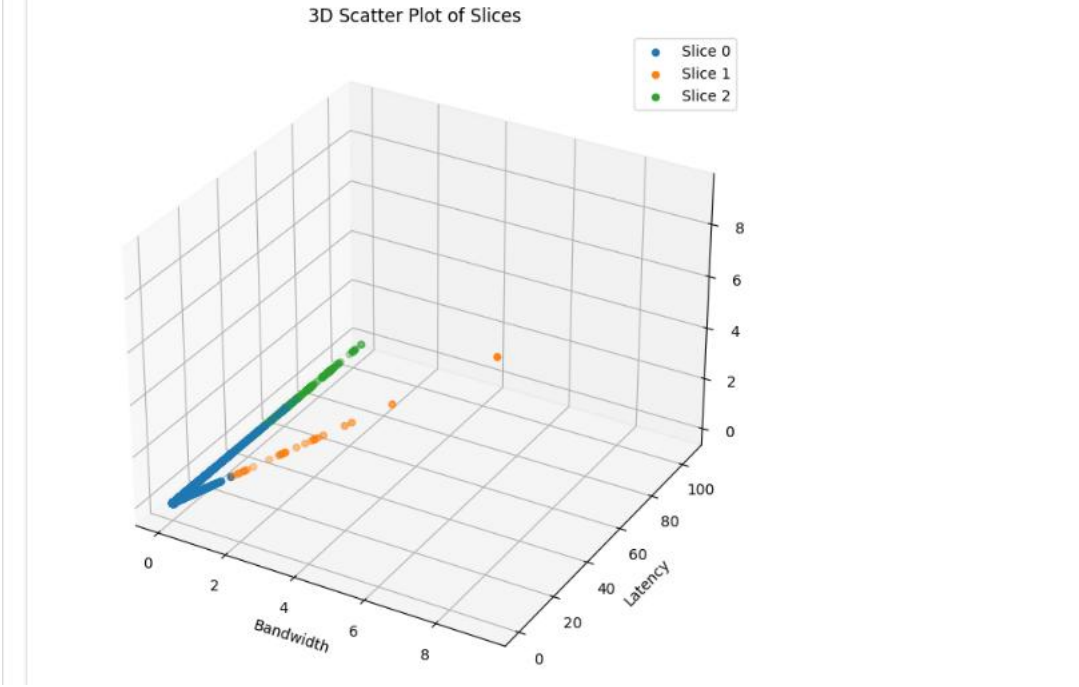


Figure 8.7.1.2 Visualising With 3D Scatter Plot

Figure 8.7.1.3 uses heat map to visualise and validate the performance of the model's prediction. The confusion matrix provides the true positive, false positive, true negative and the false negative.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Assuming 'y_test' and 'y_pred' are available
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False, square=True)
plt.xlabel('Predicted Slice')
plt.ylabel('True Slice')
plt.title('Confusion Matrix')
plt.show()
```

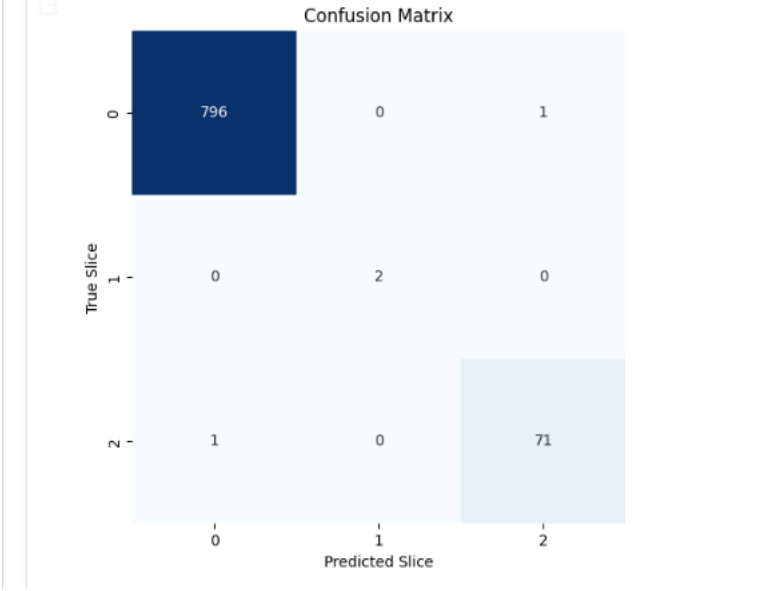


Figure 8.7.1.3 Confusion Matrix - Part1 Dataset

The classification report in Figure 8.7.1.4 further evaluates the performance of the proposed model pipeline by providing values for precision, recall, F1-score, and support for each orchestration decision. In this context, support refers to the number of actual instances of each class in the dataset, indicating how frequently each category appears. This helps in assessing the reliability of performance metrics, as a class with very low support might lead to less meaningful precision and recall scores. Based on this report, the model performs efficiently across different orchestration scenarios. The diagonal values from top-left to bottom-right shows correct matches with orchestrating the best slice for each application. The other values in the matrix represents misclassification. From the confusion matrix in Figure 8.7.1.3 we can see the following for each slice:

Slice 0: 796 applications out of 797 applications that slice 0 got orchestrated for were perfectly matched. 1 slice was allocated to slice 2. This was also an alternative slice suggested by the model that required lesser network resource and functions than slice 0.

Slice 1: All the applications allocated to slice 1 which were 2 applications were perfectly matched.

Slice 2: 71 applications out of the 72 applications that slice 2 got orchestrated for were perfectly matched. 1 slice was allocated to slice 0 which was shown as the alternative best fit in the result.

This validates the 0.998 accuracy result from Figure 8.7.1.1.

```
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, precision_score,confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score,KFold

features = part1[['Bandwidth', 'Latency', 'Throughput', 'DataRate', 'Jitter', 'Packet_Loss', 'Data_Volume', 'Speed', 'Packet_Delay', 'Reliability']]
# Standardize the features
scaler = StandardScaler()
scaled_features = pd.DataFrame(scaler.fit_transform(features),columns=['Bandwidth', 'Latency', 'Throughput', 'DataRate', 'Jitter', 'Packet_Loss', 'Data_Volume', 'Speed', 'Packet_Delay', 'Reliability'])
scaled_features['ProtocolName']= LabelEncoder().fit_transform(part1['ProtocolName'])
scaled_features['Source.IP']= LabelEncoder().fit_transform(part1['Source.IP'])

km = KMeans(n_clusters= 4, init='random',n_init=10, random_state=0)
km.fit(scaled_features)
part1['klabel'] = km.labels_

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[796   0   1]
 [  0   2   0]
 [  1   0  71]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       797
           1       1.00      1.00      1.00         2
           2       0.99      0.99      0.99        72

    accuracy                           1.00       871
   macro avg       0.99      0.99      0.99       871
weighted avg       1.00      1.00      1.00       871
```

Figure 8.7.1.4 Classification Report - Part1 Dataset

The classification report on our first dataset, part 1 has shown very high precision, support, f1-score and recall with a 100% (1.0) accuracy. The macro average had a very high score of 0.99 all through and the weighted average has a very high score of 1.0 indicating that the model performs exceptionally well and indicating a balanced performance across all slice classes.

$$Precision = \frac{\text{number of true positive predictions}}{\text{total number of positive predictions}}$$

The high precision achieved from validating the model with part1 dataset indicates few false negatives.

$$Recall = \frac{\text{number of true positive predictions}}{total\ number\ of\ actual\ positives}$$

The high recall which is also the sensitivity of the model achieved from validating the model with part1 dataset indicates few false negatives.

$$F1 - Score := \frac{2 * (\text{Precision} * \text{Recall})}{Precision + Recall}$$

The harmonic mean of the precision and recall which is the F1-Score provides a balance between precision and recall. The results indicates a highly well balanced model.

$$Accuracy := \frac{\text{number of correct predictions}}{Total\ number\ of\ predictions}$$

This shows a high value of 1.0 against part1 of the dataset.

$$Macro\ Average = \frac{1}{N}\sum_{i=1}^{N}(Metric_i)$$

N is the number of slices and the average of precision, F1-score and recall across all the slices allocated to all the applications in part1 dataset has a very high macro average score.

$$Weighted\ Average = \sum_{i=1}^{N} (W_i . Metric_i)$$

W_i is the proportion of true instances of slice 1 among all the slices. The weighted average score resulted in a high score of 1.

8.7.2 Part2 Dataset

The validation of the model was evaluated with another set of datasets to evaluate its behaviour when subjected to a different input. This dataset is labelled part2. Similar tests carried out on part1 dataset were carried out on part2. Figure 8.7.2.1 shows the accuracy of the classification for part2 dataset.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


X_imputed = imputer.transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, part2['Cluster'], test_size=0.2, random_state=42)

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Predict the clusters for the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9954128440366973
```

Figure 8.7.2.1 Machine Learning Classification Accuracy - Part2 Dataset

The accuracy of the model's classification for orchestrating slices to applications using Random Forest algorithm against the part2 dataset shows a very high accuracy of 0.995 which indicates the model performs well.

The classification report and confusion matrix for the proposed model by evaluating it against part2 dataset is shown in Figure 8.7.2.2.

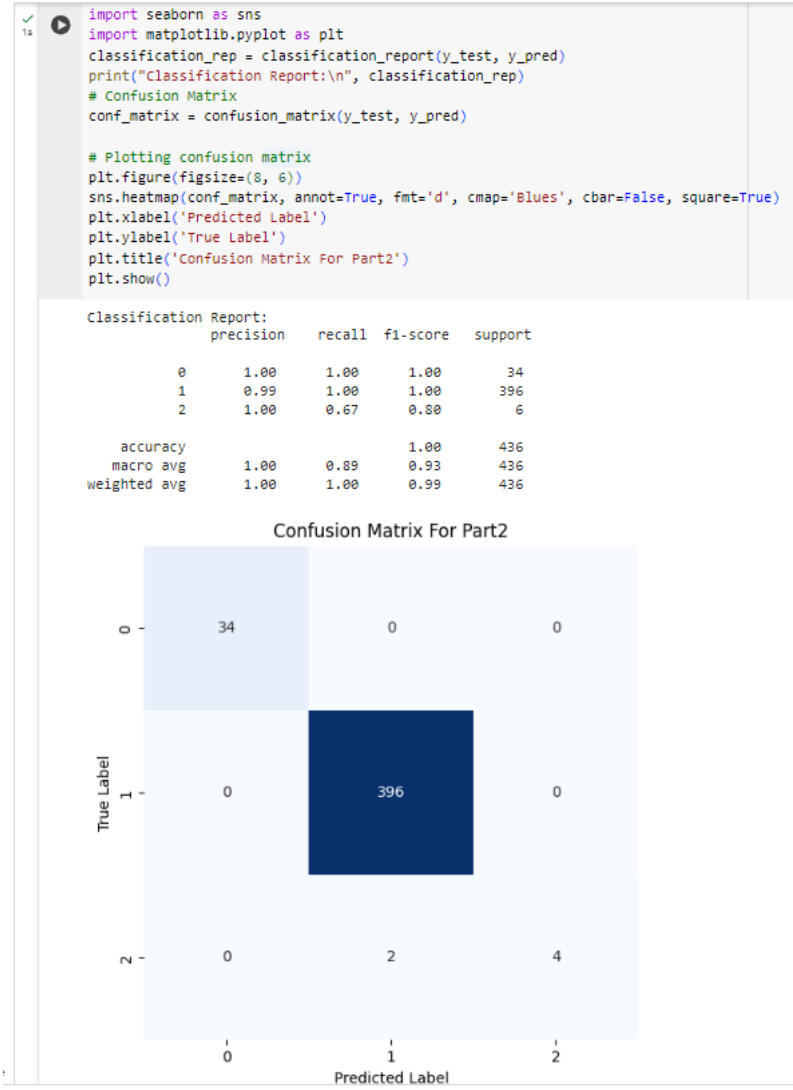


Figure 8.7.2.2 Classification Report and Confusion Matrix - Part2 Dataset

The classification report in Figure 8.7.2.2 further shows the performance of the proposed model pipeline by providing the values for support, recall, F1-score and precision of each orchestration. Based on this report, the model performs efficiently. The diagonal values from top-left to bottom-right shows correct matches with orchestrating the best slice for each application. The other values in the matrix represents misclassification. Form the confusion matrix in Figure 8.7.2.2 we can see the following for each slice:

Slice 0: All the applications allocated to slice 0 which were a total of 34 applications were perfectly matched.

Slice 1: 396 applications out of 398 applications that slice 1 got orchestrated for were perfectly matched. 2 slices were allocated to slice 2. This was also an alternative slice suggested by the model that required lesser network resource and functions than slice 1 and was shown as the alternative best fit in the result.

Slice 2: All the applications allocated to slice 2 which were a total of 4 applications were perfectly matched.

This validates the 0.995 accuracy result from Figure 8.7.1.1.

The classification report on our second dataset, part 2 has shown very high precision, support, f1-score and recall with a 100% (1.0) accuracy. The macro average had a very high score of average of 0.99 all through and the weighted average has a very high score of average of 0.99 indicating that the model performs exceptionally well and indicating a balanced performance across all slice classes.

8.7.3 Part3 Dataset

Part1 and Part2 has shown that the proposed model has high performance and accuracy. The model was further validated with a different input which is the part3 dataset to compare result and verify the impact of varying inputs on the proposed model pipeline. This dataset is labelled part3. Similar tests carried out on part1 and part2 dataset were carried out on part3 dataset. Figure 8.7.3.1 shows the accuracy of the machine learning algorithm classification for part3 dataset.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


X_imputed = imputer.transform(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, part3['Cluster'], test_size=0.2, random_state=42)

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Predict the clusters for the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9954128440366973
```

Figure 8.7.3.1 Machine Learning Classification Accuracy - Part3 Dataset

The accuracy of the model's classification for orchestrating slices to applications using Random Forest algorithm against the part3 dataset shows a very high accuracy of 0.995 which indicates the model performs well.

The classification report and confusion matrix for the proposed model by evaluating it against part3 dataset is shown in Figure 8.7.3.2.

```
print("Classification Report:\n", classification_rep)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False, square=True)
plt.xlabel('Predicted Slice')
plt.ylabel('True Slice')
plt.title('Confusion Matrix For Part3 Dataset')
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       406
           2       1.00      0.93      0.97        30

    accuracy                           1.00       436
   macro avg       1.00      0.97      0.98       436
weighted avg       1.00      1.00      1.00       436
```

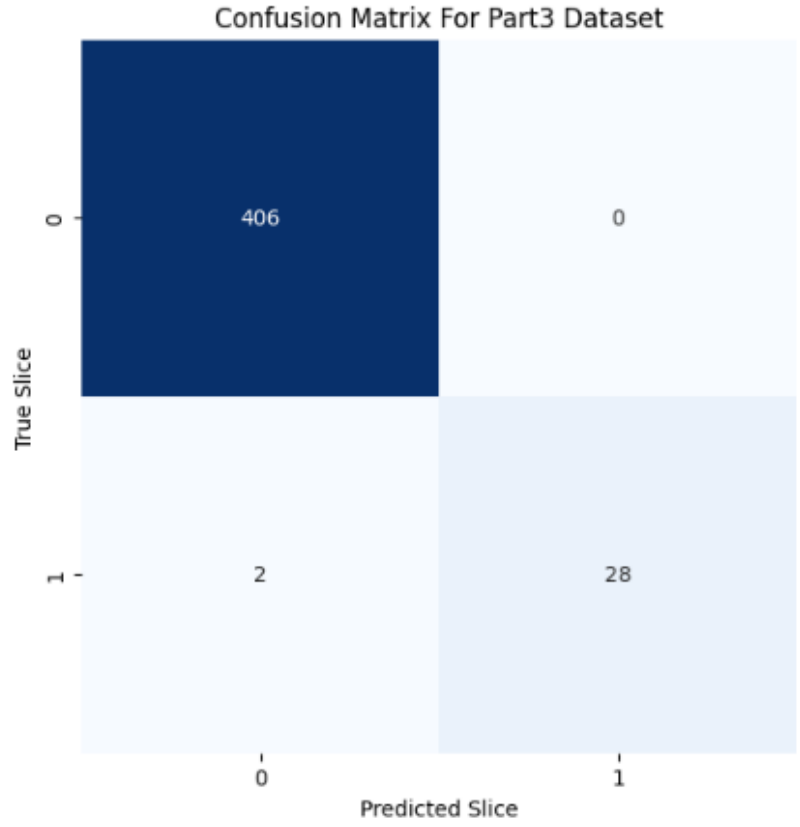


Figure 8.7.3.2 Classification Report and Confusion Matrix - Part3 Dataset

Cross validation was also used to validate the proposed model's generalisation capability and its performance on various subsets of the data. The scikit-learn library was used to perform the cross-

validation with K-fold and cross_val_score to obtain the evaluation score for each fold. Figure 8.7.3.3 shows the result of the cross-validation and Figure 8.7.3.4 represents the results visually with a box plot. The box plot shows the score distribution across different folds.

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer

# Assuming you have your features (X) and labels (y) ready

# Create a RandomForestClassifier (or any other model)
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Define the number of splits for KFold
num_splits = 5  # You can adjust this based on your preference

# Create a KFold object
kf = KFold(n_splits=num_splits, shuffle=True, random_state=42)

# Define a scoring metric (you can choose based on your task, e.g., accuracy, precision, etc.)
scoring_metric = 'accuracy'

# Use cross_val_score for cross-validation
cross_val_results = cross_val_score(model, X_train, y_train, cv=kf, scoring=scoring_metric)

# Print the cross-validation results
print(f'Cross-Validation Results: {cross_val_results}')
print(f'Mean Accuracy: {cross_val_results.mean()}')
```

```
Cross-Validation Results: [0.99425287 1.         0.99712644 0.99137931 0.99425287]
Mean Accuracy: 0.9954022988505746
```

Figure 8.7.3.3 Cross Validation Of Model Using Part3 Dataset

The cross-validation result shows the accuracy scores obtained from various subset of the training data and the resulting scores are very high. The scores are ranging from 99.14% to 100% which indicates a high performing model with high accuracy when allocating slice and predicting with new and unseen data. The consolidated mean accuracy score is very high, giving a result of 99.5% which indicates a very high performance of the overall model.

The box plot in Figure 8.7.3.4 shows the interquartile range of the accuracy scores. From the box plot, it is shown that the minimum accuracy value is 0992 and the maximum is 1.0 with the model's accuracy ranging between 0.995 and 0.997.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Visualize cross-validation results with a box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x=cross_val_results)
plt.title('Cross-Validation Results')
plt.xlabel('Accuracy')
plt.show()
```

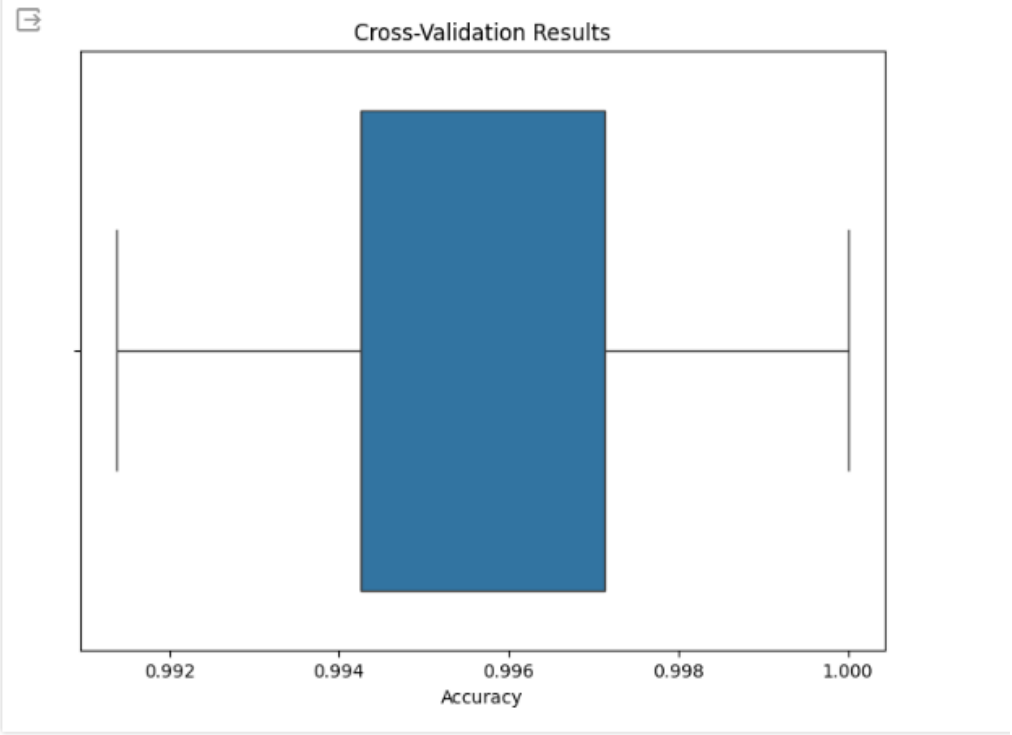


Figure 8.7.3.4 Box Plot Cross Validation of Model Using Part3 Dataset

8.7.5 Result

A summary of the results from the part 1, part 2 and part 3 of the dataset is shown in Table 8.7.8.1.

Table 8.7.8.1. Summary Model Accuracy

	Part1	Part2	Part3
RF Classification Accuracy	0.998	0.995	0.995
NWDAF (Mean Squared Error)	0.22		
Classification Report Accuracy	1.0	1.0	1.0
Weighted Average	1.0	0.99	1.0
Precision	0.99	0.99	1.0

The results show the proposed framework with its ML-based eSS model pipeline performs well with very high accuracy and can efficiently orchestrate slice for new unseen data improving its suitability for cross-domain application. It effectively manages network resources by also effectively orchestrating alternative slice to maintain good user experience and improve the slice QoS.

It can also be shown from the scatter plot in Figure 8.7.5.1 that the proposed model in this research's predicted slices align well with the actual slice allocation to each group of applications. The diagram shows points of the same colour close to each other which visually indicates that the predicted allocations are consistent with the actual slice allocations in the feature space. Each point in the scatter plot represents an application in the part1 network traffic dataset.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', label='Predicted Slice')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, marker='x', cmap='viridis', label='Actual Slice', alpha=0.5)
plt.xlabel('Bandwidth')
plt.ylabel('Latency')
plt.title('Actual vs. Predicted Slices')
plt.legend()
plt.show()
```

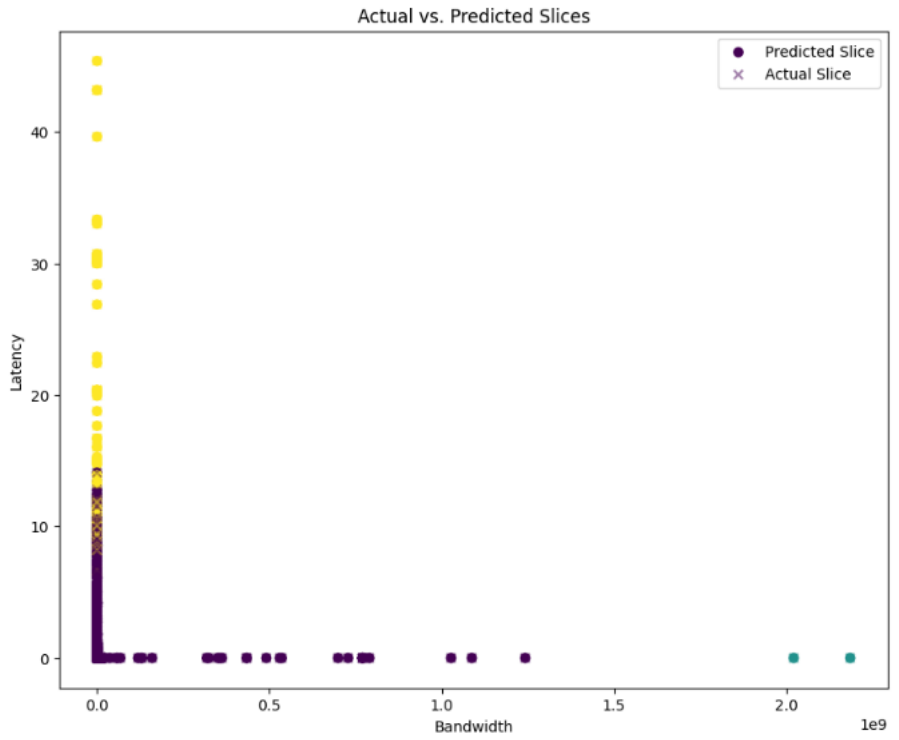


Figure 8.7.5.1 Predicted Vs Actual - Part1 Dataset

8.8 Resource Utilisation

We compare the performance of resource utilisation in our proposed framework against two benchmarks: static slicing and heuristic based orchestration.

Static Slicing: The traditional static network resource allocation is allocated based on pre-defined configuration without any adjustment. Standard network slices are predefined, configured and allocated as specified. This approach is commonly used in existing networks.

Heuristic-based Orchestration: A reactive scaling and allocation of network resources based on observed demands, without any learning components or prediction. Recent studies [84] have shown that some network slicing frameworks employ this rule-based orchestration techniques benchmark.

8.8.1 Result

The simulation result shows that our proposed ML-based framework not only achieved better throughput and latency when compared to benchmarks, but it also achieves better resource utilisation as illustrated in Figure 8.8.1.

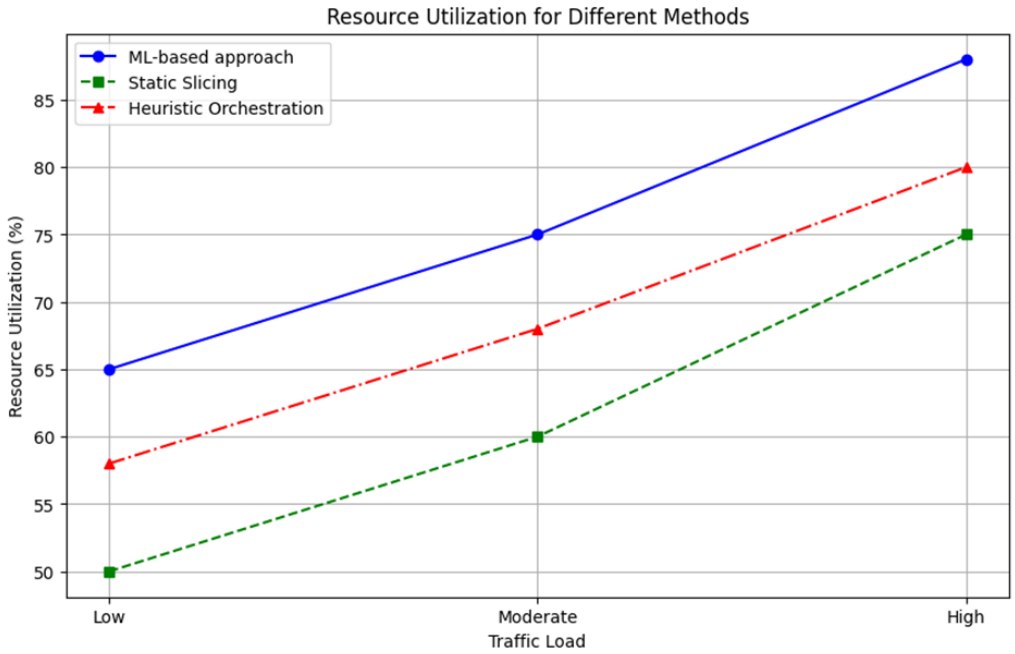


Figure. 8.8.1: Plot showing resource utilisation for different methods

Our ML-based framework avoids over-provisioning and efficiently manages resources by proactively allocating network slice based on the varying demands while the static slicing approach has a lower performance as shown in the result due to its inability to adapt to the varying demands. The heuristic method lags in responsiveness which degrades its performance.

From the simulation result, an improvement of 15-29% was achieved by our proposed ML-based orchestration framework in resource utilisation compared to static slicing and an improvement of 8-18% was achieved compared to heuristic-based orchestration.

8.9 Summary

Chapter 8.3 evaluated the metrics adopted in this study by analysing the network functions and resources allocated for each type of orchestrated slice in the eSS model.

Chapter 8.4 validated the number of slices required by evaluating using various methods such as elbow method, silhouette score, Davies-Bouldin index and showing the results in Table 8.4.1.

Chapter 8.5 validated the slice allocation and the alternative slice in the event of limited network resource, and it was shown that a slice with lesser network resource and function requirement was proposed by the model to meet the application requirement.

Chapter 8.6 validated the accuracy of the output of the LazyPredict which was Random Forest to predict the slice orchestration. Various diagrams and methods such as classification report, confusion matrix, accuracy score and precision score were used to test and validate the model.

Chapter 8.7 validated the efficiency of the eSS model by varying the dataset input to emulate a dynamic environment of varying requirements. The accuracy of the slice allocation was measured for each part of dataset fed into the model.

Chapter 8.8 presented the simulations results, analysed, and compared them against both the heuristic-based orchestration and static slicing benchmarks to validate the performance of resource utilisation in the eSS model.

9.0 Conclusion and Future Work

9.1 Summary of Contributions

This research addressed the critical challenge of efficiently orchestrating network slices in 5G networks through the development of a novel machine learning-based model and an overarching adaptive framework. This work was driven by the need for more efficient, adaptive, and scalable network slicing solutions to meet the dynamic and heterogeneous requirements of next-generation applications by effectively managing and allocating the limited network resource blocks. This initiated the research questions that guided this study. The questions were centred around how we could efficiently orchestrate a 5G network slice by subslicing them to reduce waste and efficiently allocate network. This then led to the to the development of an adaptive framework for managing end-to-end slicing in 5G networks. This framework was complemented by a machine learning pipeline model, implemented using the LazyPredict library, which aimed to optimise the slice orchestration process by subslicing a network slice.

9.2 Key Findings and Contributions

The key contributions of this thesis are as follows:

9.2.1. Proposed Adaptive Framework: The ONSSO framework was successfully developed to structure and manage the orchestration of network slices end-to-end efficiently, introducing key components such as resource allocation, monitoring, caching, and automated decision-making processes. The proposed framework outperforms the conventional static and heuristic-based methods across key metrics such as latency, throughput, and resource utilisation as demonstrated in Section 8.8. This is shown by its ability to manage high load and dynamically adjust proactively to varying complex demands effectively. The framework enables integration with AI-based tools for improved agility and scalability. This framework demonstrated significant improvements in orchestration efficiency compared to existing solutions.

9.2.2. Machine Learning Model (eSS): A series of machine learning algorithms were chained and implemented under the proposed model eSS to optimise slice orchestration decisions, focusing on predictive resource allocation and real-time adjustments. Our approach dynamically adjusts slices by sub-slicing and service chaining to orchestrate a bespoke network slice on demand. Supervised learning by automating the selection of best-fit machine learning algorithm was leveraged upon for accurate feature selection and network functions classifications to form a network slice. Reinforcement learning was leveraged upon for proactively learning and forecasting the network resource required by time. This model dynamically adjusts slice resources to match time-varying demands. The algorithms demonstrated their ability to outperform traditional rule-based orchestration methods by adapting to network changes proactively. The application of machine learning, specifically using the LazyPredict library, proved effective in enhancing the slice orchestration process. The model showed capability in predicting better slice configurations and resource allocations. The simulation results clearly demonstrate the effectiveness of the proposed ML-based framework for efficiently orchestrating 5G network slice as demonstrated in Section 8.4.

9.2.3. A New Vertical (CNaaS): The adaptive framework ONSSO enabled a smart vertical called CNaaS which is like a cloud service that scales up and down as required. It facilitates the collection of network traffic data, analysing the user devices on the network needs, configuring the required subslice templates, orchestrating and allocating the resource to orchestrated subslice. The simulated

result compared with results from existing studies shows the high accuracy of the chosen model in this research and scalability of the CNaaS framework.

9.2.4. Streamlined Mobile Network Operators Collaborative Process: The real time availability of data to mobile network operators via the message bus within the ONSSO framework enables automation of network resource allocation through to the mobile network operators. It reduces latency and removes the need for mobile network operators to wait for information to be sent to them before they can make predictions or allocate resources.

9.2.5. Integrated Auto-Selection Best Fit Machine Learning Model: The integration of the LazyPredict python module to automatically select the best machine learning algorithm for classification based on the dynamic dataset available in real time and for proactive predictions is an enabler to the efficient allocation of network resources to a 5G network subslice. It allows only required network functions and network resources to be chained. It estimates the optimised number and types of subslice to be orchestrated over a given network, The optimised model was tested against an available dataset and the best machine learning algorithm for the network slice orchestration was tested and selected after analysing and comparing against an integrated series of twenty-nine different machine learning models.

9.2.6. Performance Evaluation: Extensive simulations and real-world testbeds were utilised to evaluate the performance of the proposed solution. Using Python and MATLAB simulations, we were able to validate the effectiveness of our proposed framework and model. Results indicated that eSS reduces resource wastage, minimises latency, and enhances Quality of Service (QoS) for end-users compared to existing solutions. The use of both the historical dataset and an augmented dataset generated using AI techniques provided robust testing scenarios.

9.2.7. Scalability and Adaptability: This research confirmed that the proposed framework and model scale effectively with increasing network complexity and adapt to a variety of traffic patterns, service types, and user demands, which are critical for the future success of 5G and beyond as demonstrated in Section 8.7.

9.2.8. Efficiency Metrics: Through our research, we identified and validated a set of key metrics that can be used to measure and improve the efficiency of network slice orchestration. These metrics provide a quantitative basis for evaluating orchestration performance.

9.2.9 Machine Learning Based Pipeline Major Steps: Network slicing enables the sharing of physical resources into multiple virtual networks which serve specific requirements, thereby efficiently utilising and sharing the physical resources across application services [58]. This research has further enhanced the use sharing and reuse of network slices by intelligently and dynamically sub-slicing a network slice to reduce capital expenditure and manage the limited available network resources between application services. Thereby making the model flexible and scalable.

It can be deduced from the findings of this research that the following steps are required to dynamically sense network traffic and efficiently orchestrate the network slices:

1. Traffic flow analysis.
2. Group network characteristics using unsupervised learning. Random Forest and KNN were chosen.
3. Group application to assign to grouped features – Classification.

4. Naïve Bayes was used to calculate the probability that new app entering the network belongs to a cluster.
5. Reinforcement learning model is used to continually learn, predict and improve network performance and resource allocation efficiency.

9.3 Implications of the Findings

The findings from this research have significant implications for both academic and industrial domains of network slice orchestration:

9.3.1 Network Efficiency: By implementing the Reinforcement Learning predictive models based on machine learning, the orchestrator can make more informed decisions, improving resource efficiency and ensuring that critical subslices (such as those for Ultra-Reliable Low-Latency Communication, or URLLC) receive appropriate prioritisation and only required network functions without over-provisioning resources. The ability of the framework to orchestrate alternative best fit slices reduces the risk of denial and improves users QoE.

9.3.2. Operator Flexibility: The ONSSO framework enhances the availability of user equipment requirement information to network operators as well as enhances the flexibility of network operators to deploy, manage, and adjust slices in real-time, offering faster adaptation to evolving customer needs and varying service requirements. This available information in real-time and flexibility will be key to supporting services such as autonomous vehicles, large-scale IoT deployments and virtual/augmented reality,

9.3.3. Cost Reduction: The proposed approach has the potential to reduce Capital expenditures (CapEx) and operating expenses (OpEx) by minimizing manual interventions and improving resource allocation efficiency. The proposed ONSSO framework and eSS model contribute to lowering the operational costs for network operators while maintaining high service quality. Streamlining the collaborative process renders existing part of the network operator process cycle redundant due to access to real-time network slice requirement data. This study demonstrates the enhanced sharing of the available network resources by sub-slicing the formulated network slice. The three standard network slices; enhanced mobile broadband (eMBB), massive machine type communication (mMTC) and ultra-reliable low latency communication (URLLC) are static [47] and could result into waste of resources due to over allocating required resources, but this research has shown the opportunity of reducing capital expenditure by efficiently orchestrating resources based specifically on applications requirements.

9.3.4. AI in Telecommunications: The success of eSS highlights the growing role of artificial intelligence in telecommunications, particularly in dynamic, data-driven environments like 5G networks and beyond. It underscores the potential for AI models to revolutionise traditional network management and orchestration tasks.

9.4 Limitations and Future Work

While this thesis presents a robust approach to improving network slice orchestration with resource allocation, there are limitations that should be acknowledged and future research directions proposed:

1. Generalisability Across Networks: The framework and model were tested on a limited set of live 5G network architectures although varying environments parameters were simulated to

validate these. Though the results are promising, future studies should test ONSSO and eSS across a broader range of live 5G network configurations, including different infrastructure setups and varying deployment scales. Testing the framework and model at scale in live 5G networks was beyond the scope of this study. The performance under extreme scalability conditions remains to be fully explored.

2. Security Considerations: This work has primarily focused on performance improvements in orchestration; however, the security of network slice orchestration, especially in scenarios involving malicious attacks or data breaches, remains an open issue. Future research should address these security challenges to ensure that the proposed solution remains resilient to threats. Development of mechanisms to enhance the cyber-resilience of the orchestration framework, ensuring robust performance even under potential security threats or attacks could be further studied.
3. Dataset Limitations: Although we used both real (Unicauca Dataset) and augmented datasets, the generalisability of our results to all possible network scenarios may be limited. Future work could benefit from more diverse and extensive real-world datasets
4. Energy Efficiency: Incorporate energy consumption as a key metric and optimisation target in the orchestration process, aligning with growing sustainability concerns in the telecom industry
5. Exploring the integration of the proposed framework with emerging technologies such as Blockchain and AI including investigating the potential challenges and synergies will further enhance this study and the use of CNaaS across all domains.
6. Extending the auto-selection algorithm (LazyPredict) used to include other neural networks in future studies will enhance the robustness and scalability of the proposed eSS

9.5 Concluding Remarks

This thesis has presented a novel approach to addressing the critical challenge of efficient network slice orchestration in 5G networks. The proposed eSS model and adaptive ONSSO framework represent a step forward in autonomous network management and provide a foundation for future research and development in this area. Significant improvements in orchestration efficiency has been demonstrated and the identified metrics as well as the experimental results provide a solid foundation for future research and development in this rapidly evolving field.

As the telecommunications industry evolves towards more complex and service-oriented architectures, the need for intelligent, efficient, adaptive orchestration solutions will continue to grow. This research offers a promising path forward, contributing to the ongoing evolution of next-generation mobile networks and the realisation of their full potential across diverse use cases and robustness of future network ecosystems.

References

[1] Zhang, H., Liu, N., Chu, X., Long, K., Aghvami, A.H. and Leung, V.C.M., 2017. Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges. IEEE Communications Magazine, 55(8), pp.138-145.

[2] Jia, Q., et al., 2019. Caching resource sharing for network slicing in 5G core network: A game theoretic approach. Journal of Organizational and End User Computing (JOEUC), 31(4).

[3] Xu, X., Zhang, H., Dai, X., Hou, Y., Tao, X. and Zhang, P., 2014. SDN based next-generation mobile network with service slicing and trials. China Communications, 11(2), pp.65-77.

[4] Heinonen, J., Korja, P., Partti, T., Flinck, H. and Pöyhönen, P., 2016. Mobility management enhancements for 5G low latency services. In: 2016 IEEE International Conference on Communications Workshops (ICC), Kuala Lumpur, Malaysia.

[5] Next Generation Mobile Network Alliance, 2015. NGMN 5G white paper.

[6] 3rd Generation Partnership Project (3GPP), 2015. Policy and charging control architecture (Release 13). Technical Specification Group Services and System Aspects.

[7] Nikaein, N., Schiller, E., Favraud, R., Katsalis, K., Stavropoulos, D. and Alyafawi, I., 2015. Network store: Exploring slicing in future 5G networks. In: Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture.

[8] 3rd Generation Partnership Project (3GPP), 2015. Feasibility study on control and user plane separation of EPC nodes.

[9] Zhang, H., Liu, N., Chu, X., Long, K., Aghvami, A.H. and Leung, V.C.M., 2014. Resource allocation in spectrum-sharing OFDMA femtocells with heterogeneous services. IEEE Transactions on Communications, 62(7), pp.2366-2377.

[10] López-Pérez, D., Valcarce, A., de la Roche, G. and Zhang, J., 2009. OFDMA femtocells: A roadmap on interference avoidance. IEEE Communications Magazine, 47(9), pp.41-48.

[11] Yun, J.H. and Shin, K.G., 2011. Adaptive interference management of OFDMA femtocells for co-channel deployment. IEEE Journal on Selected Areas in Communications, 29(6), pp.1225-1241.

[12] Yazıcı, V., Kozat, U.C. and Sunay, M.O., 2014. A new control plane for 5G network architecture with a case study on unified handoff, mobility, and routing management. IEEE Communications Magazine, 52(11), pp.76-85.

[13] 3rd Generation Partnership Project (3GPP), 2020. About 3GPP. Available at: [https://www.3gpp.org/about-3gpp] [Accessed 04 December 2020].

[14] 3rd Generation Partnership Project (3GPP), 2007. 3GPP scope and objectives. Approved by 3GPP organizational partners by correspondence.

[15] Patwary, M.N., et al., 2020. Potential short- and long-term disruptions and transformative impacts of 5G and beyond wireless networks.

[16] Singh, S.K., Salim, M.M., Cha, J., Pan, Y. and Park, J.H., 2020. Machine learning-based network sub-slicing framework in a sustainable 5G environment.

[17] Arfaoui, G., et al., 2018. A security architecture for 5G networks. IEEE Access, 6, pp.22466-22479.

[18] Young, P.R. and Sharma, S., 2011. Virtualization: A review and future directions executive overview. American Journal of Information Technology, 1, pp.1-37.

[19] Simon, C. and David, B., 2006. The virtualization reality. ACM Queue, 4, pp.34-41. DOI: 10.1145/1189276.1189289.

[20] Gupta, A. and Jha, R.K., 2015. A survey of 5G network: Architecture and emerging technologies. IEEE Access, 3, pp.1206-1232.

[21] Agyapong, P., Iwamura, M., Staehle, D., Kiess, W. and Benjebbour, A., 2014. Design considerations for a 5G network architecture. IEEE Communications Magazine, 52(11), pp.65-75.

[22] Hernández-Chulde, C. and Cervelló-Pastor, C., 2019. Intelligent optimization and machine learning for 5G network control and management. In: De La Prieta, F., et al. (eds) Highlights of Practical Applications of Survivable Agents and Multi-Agent Systems. The PAAMS Collection, PAAMS 2019. Communications in Computer and Information Science, vol. 1047. Springer, Cham.

[23] Kafle, V.P., Fukushima, Y., Martinez-Julia, P. and Miyazawa, T., 2018. Consideration on automation of 5G network slicing with machine learning. In: 2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K), Santa Fe, Argentina, pp.1-8.

[24] Iqbal, M. and Zhu, Y. Supervised machine learning approaches: A survey.

[25] Thantharate, A., Paropkari, R., Walunj, V. and Beard, C., 2019. DeepSlice: A deep learning approach towards an efficient and reliable network slicing in 5G networks. In: 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, pp.0762-0767.

[26] Le, L., Lin, B.P., Tung, L. and Sinh, D., 2018. SDN/NFV, machine learning, and big data driven network slicing for 5G. In: 2018 IEEE 5G World Forum (5GWF), Silicon Valley, CA, USA, pp.20-25.

[27] Abidi, M.H., Alkhalefah, H., Moiduddin, K., Alazab, M., Khan Mohammed, M., Ameen, W. and Gadekallu, T.R., 2021. Optimal 5G network slicing using machine learning and deep learning concepts. Computer Standards & Interfaces, 76, p.103518.

[28] Zanzi, L., Sciancalepore, V., Garcia-Saavedra, A. and Costa-Pérez, X., 2018. OVNES: Demonstrating 5G network slicing overbooking on real deployments.

[29] Hantouti, H., Benamar, N. and Taleb, T., 2020. Service function chaining in 5G & beyond networks: Challenges and open research issues. IEEE Network, 34(4), pp.320-327.

[30] Saunders, M., Lewis, P. and Thornhill, A., 2009. Research methods for business students. Pearson, New York, p.45. [31] Zhou, K., Liu, T. and Zhou, L., 2015. Industry 4.0: Towards future industrial opportunities and challenges. In: 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp.2147-2152.

[31] Mdpi, (2023) 'Efficient Network Slicing with SDN and Heuristic Algorithm for Low Latency Services in 5G Networks', Sensors, 23(13), p. 6053.

[32] Pries, R., Morper, H., Galambosi, N. and Jarschel, M., 2016. Network as a service - A demo on 5G network slicing. In: 2016 28th International Teletraffic Congress (ITC 28), pp.209-211. DOI: 10.1109/ITC-28.2016.136.

[34] Su, R., et al., 2019. Resource allocation for network slicing in 5G telecommunication networks: A survey of principles and models. IEEE Network, 33(6), pp.172-179.

[35] Ravindran, S., Chaudhuri, S., Bapat, J. and Das, D., 2019. Isolation-based subslices for throughput optimization in 5G radio access network. In: 2019 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), pp.1-6.

[36] Ravindran, S., Chaudhuri, S., Bapat, J. and Das, D., 2020. EESO: Energy efficient system-resource optimization of multi-subslice-connected user in 5G RAN. In: 2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), pp.1-6.

[37] Ravindran, S., Chaudhuri, S., Bapat, J. and Das, D., 2019. Required delay-based network subslices resource optimization for 5G radio access network. In: 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), pp.1-6.

[38] Kaloxylos, A., 2018. A survey and an analysis of network slicing in 5G networks. IEEE Communications Standards Magazine, 2(1), pp.60-65.

[39] Malandrino, F., Chiasserini, C., Casetti, C., Landi, G. and Capitani, M., 2019. An optimization-enhanced MANO for energy-efficient 5G networks. IEEE/ACM Transactions on Networking, 27, pp.1756-1769.

[40] Machine Learning Mastery, 2024. Dimensionality reduction algorithms with Python. Available at: https://machinelearningmastery.com/dimensionality-reduction-algorithms-with-python/.

[41] Jackson, D.A., 1993. Stopping rules in principal components analysis: A comparison of heuristical and statistical approaches. Ecology, 74(8), pp.2204-2214.

[42] Unicauca Dataset, 2018. Google Drive. Available at: https://drive.google.com/drive/folders/1FcnKUlSqRb4q5PkGfAGHz-g7bVKL8jmu?usp=sharing.

[43] Rojas Meléndez, J., Rendón, A. and Corrales, J., 2018. Personalized service degradation policies on OTT applications based on the consumption behavior of users. In: Lecture Notes in Computer Science, pp.543-557. DOI: 10.1007/978-3-319-95168-3_37.

[44] L'heureux, A., Grolinger, K., Elyamany, H.F. and Capretz, M.A., 2017. Machine learning with big data: Challenges and approaches. IEEE Access, 5, pp.7776-7797.

[45] Salhab, N., Langar, R. and Rahim, R., 2021. 5G network slices resource orchestration using machine learning techniques. Computer Networks, 188, p.107829.

[46] 3GPP, 2018. TR 28.801: Study on management and orchestration of network slicing for next generation network (Release 15).

[47] Li, R., Zhao, Z., Zhou, X., Ding, G., Chen, Y., Wang, Z. and Zhang, H., 2017. Intelligent 5G: When cellular networks meet artificial intelligence. IEEE Wireless Communications, 24(5), pp.175-183.

[48] Sallent, O., Pérez-Romero, J., Ferrús, R. et al., 2017. On radio access network slicing from a radio resource management perspective. IEEE Wireless Communications, 24(5), pp.166-174. DOI: 10.1109/MWC.2017.1600220.

[49] Han, B., Feng, D., Ji, L. et al., 2017. A profit-maximizing strategy of network resource management for 5G tenant slices. [EB/OL]. Available at: [Link needed] (Accessed: 26 September 2017).

[50] Caballero, P., Banchs, A., De Veciana, G. et al., 2018. Network slicing for guaranteed rate services: Admission control and resource allocation games. IEEE Transactions on Wireless Communications, 17(10), pp.6419-6432. DOI: 10.1109/TWC.2018.2859918.

[51] Han, B., Feng, D., ScianCALEPORE, V. et al., 2018. Rational impatience admission control in 5G-sliced networks: Shall I bide my slice opportunity? [EB/OL]. Available at: [Link needed] (Accessed: 31 October 2018).

[52] Wu, D., Zhang, Z., Wu, S. et al., 2018. Biologically inspired resource allocation for network slices in 5G-enabled Internet of Things. IEEE Internet of Things Journal.

[53] ScianCALEPORE, V., Di Renzo, M. and Costa-Pérez, X., 2019. STORNS: Stochastic radio access network slicing. In: 2019 IEEE International Conference on Communications (ICC), Shanghai, China, pp.1-7.

[54] Boutaba, R., Salahuddin, M.A., Limam, N. et al., 2018. A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. Journal of Internet Services and Applications, 9(16).

[55] Internet Engineering Task Force, 2002. Architecture for describing simple network management protocol (SNMP) management frameworks.

[56] Internet Engineering Task Force, 2008. Specification of the IP Flow Information Export (IPFIX) protocol for the exchange of IP traffic flow information.

[57] Cisco Systems, 2012. Cisco IOS NetFlow Version 9. Available at: http://www.cisco.com/go/netflow. (Accessed: 14 January 2022).

[58] Li, Q.C., et al., 2016. An end-to-end network slicing framework for 5G wireless communication systems. ArXiv. Available at: https://arxiv.org/abs/1608.00572.

[59] Ahmed, A.A. and Agunsoye, G., 2021. A real-time network traffic classifier for online applications using machine learning. Algorithms, 14(250). DOI: 10.3390/a14080250.

[60] Zeng, Y., Gu, H., Wei, W. and Guo, Y., 2019. Deep-Full-Range: A deep learning-based network encrypted traffic classification and intrusion detection framework. IEEE Access, 7, pp.182-190.

[61] Kofod-Petersen, A., Heintz, F. and Langseth, H., 2011. Frontiers in Artificial Intelligence and Applications: Preface. Frontiers in Artificial Intelligence and Applications, 227.

[62] Lundberg, S.M. and Lee, S.I., 2017. A unified approach to interpreting model predictions. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S. and Garnett, R., eds. Advances in Neural Information Processing Systems 30. Curran Associates, Inc., Long Beach, CA, USA, pp. 4765–4774.

[63] Ribeiro, M.T., Singh, S. and Guestrin, C., 2016. 'Why should I trust you?' Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 13 August, pp. 1135–1144.

[64] Raschka, S., Patterson, J. and Nolet, C., 2020. Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence. Information, 11(4), Article 193.

[65] Lundberg, S.M. and Lee, S.I., 2017. A unified approach to interpreting model predictions. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. Curran Associates Inc., pp. 4768–4777.

[66] Saboorian, T. and Xiang, A., 2017. Network slicing and 3GPP service and systems aspects (SA) standard. EEE Softwarization.

[67] Anurag, T., 2019. DeepSlice & Secure5G – 5G & LTE wireless dataset. Available at: https://www.kaggle.com/anuragthantharate/deepslice?select=DeepSlice_ML_Model_Data_Public.xlsx (Accessed: Date).

[68] Rojas, J.S., 2019. IP network traffic flows labeled with 75 apps. Available at: https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps (Accessed: Date).

[69] 3GPP, 2017. TS 23.501 V0.4.0.

[70] Tikhvinskiy, V. and Bochechka, G., 2017. Quality of service in the 5G network. IEEE Communications Magazine.

[71] Abidi, A., et al., 2021. Optimal 5G network slicing using machine learning and deep learning concepts. Computer Standards & Interfaces, 76, Article 103518.

[72] Malkoc, M. and Kholidy, H.A., 2023. 5G network slicing: Analysis of multiple machine learning classifiers. arXiv preprint arXiv:2310.01747. Available at: https://arxiv.org/abs/2310.01747 (Accessed: Date).

[73] Martins, J.S.B., et al., 2023. Enhancing network slicing architectures with machine learning, security, sustainability, and experimental networks integration. IEEE Access.

[74] Bouroudi, A., et al., 2023. Dynamic machine learning algorithm selection for network slicing in beyond 5G networks. In: 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), Madrid, Spain, pp. 314-316.

[75] Shukla, N., et al., 2024. Xcelerate5G: Optimizing resource allocation strategies for 5G network using ML. In: 2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT), Greater Noida, India, pp. 417-423.

[76] Shafi, M., Molisch, A.F., Smith, P.J., Haustein, T., Zhu, P., De Silva, P. and Tufvesson, F., 2017. 5G: A tutorial overview of standards, trials, challenges, deployment, and practice. IEEE Journal on Selected Areas in Communications, 35, pp. 1201–1221.

[77] Zhang, H., Liu, N., Chu, X., Long, K., Aghvami, A.H. and Leung, V.C., 2019. Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges. IEEE Communications Magazine, 57, pp. 138–145.

[78] Ordonez-Luciano, J., Ameigeiras, P., Lopez, D., Ramos-Munoz, J.J., Lorca, J. and Folgueira, J., 2017. Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges. IEEE Communications Magazine, 55, pp. 80–87.

[79] Zhao, Y., Chen, Y., Farha, F., Li, J., Salyga, J. and Tung, Y., 2022. Hybrid machine learning for network optimisation: Opportunities and challenges. arXiv, arXiv:2205.05290.

[80] Sharma, N. and Kumar, K., 2024. A novel latency-aware resource allocation and offloading strategy with improved prioritization and DDQN for edge-enabled UDNs. IEEE Transactions on Network and Service Management, 21, pp. 1–15.

[81] Baumgartner, A., Huynh, V.P.M. and Mitschele-Thiel, A., 2020. On the optimal re-allocation of resources for sliced mobile networks. IEEE Transactions on Network and Service Management, 17, pp. 344–357.

[82] Xu, Y., Yang, K., Wang, J., Zhou, L. and Pan, M., 2020. Deep reinforcement learning for network slicing with resource allocation in 5G communication networks. IEEE Transactions on Vehicular Technology, 69, pp. 11295–11305.

[83] Zhao, Y., Chen, Y., Farha, F., Li, J., Salyga, J. and Tung, Y., 2022. Hybrid machine learning for network optimisation: Opportunities and challenges. arXiv, arXiv:2205.05290.

[84] Mdpi, (2022) ‘A Heuristic Fuzzy Based 5G Network Orchestration Framework for Network Slicing’, Applied Sciences, 12(14), p. 6942. Available at: https://www.mdpi.com/2076-3417/12/14/6942

Appendices

Appendix A: Ethics Form

UNIVERSITY OF BEDFORDSHIRE

Research Ethics Scrutiny (Postgraduate Research Students)

When completing this form please ensure that you read and comply with the following:

Researchers must demonstrate clear understanding of an engagement with the following:

1. *Integrity* - The research has been carried out in a rigorous and professional manner and due credit has been attributed to all parties involved.
2. *Plagiarism* - Proper acknowledgement has been given to the authorship of data and ideas.
3. *Conflicts of Interest* - All financial and professional conflicts of interest have been properly identified and declared.
4. *Data Handling* - The research draws upon effective record keeping, proper storage of date in line with confidentiality, statute and University policy.
5. *Ethical Procedures* - Proper consideration has been given to all ethical issues and appropriate approval sought and received from all relevant stakeholders. In addition the research should conform to professional codes of conduct where appropriate.
6. *Supervision* - Effective management and supervision of staff and student for whom the researcher(s) is/are responsible
7. *Health and Safety*- Proper training on health and safety issues has been received and completed by all involved parties. Health and safety issues have been identified and appropriate assessment and action have been undertaken.

The **Research Institutes** are responsible for ensuring that all researchers abide by the above. It is anticipated that ethical approval will be granted by each Research Institute. Each Research Institute will give guidance and approval on ethical procedures and ensure they conform to the requirements of relevant professional bodies. As such Research Institutes are required to provide the University Research Ethics Committee with details of their procedures for ensuring adherence to relevant ethical requirements. This applies to any research whether it be, or not, likely to raise ethical issues. Research proposals involving vulnerable groups; sensitive topics; groups requiring gatekeeper permission; deception or without full informed consent; use of personal/confidential information; subjects in stress, anxiety, humiliation or intrusive interventions must be referred to the University Research Ethics Committee.

Research projects involving participants in the NHS will be submitted through the NHS National Research Ethics Service (NRES). The University Research Ethics Committee will normally accept the judgement of NRES (it will never approve a proposal that has been rejected by NRES), however NRES approval will need to be verified before research can commence and the nature of the research will need to be verified.

Where work is conducted in collaboration with other institutions ethical approval by the University and the collaborating partner(s) will be required.

The **University Research Ethics Committee** is a sub-committee of the Academic Board and is chaired by a member of the Vice Chancellor's Executive Group, appointed by the ViceChancellor and includes members external to the University

Research Misconduct: Allegations of Research Misconduct against staff or post graduate (non-taught) research students should be made to the Director of Research Development.

UNIVERSITY OF BEDFORDSHIRE

Research Ethics Scrutiny (Annex to RS1 form)

SECTION A To be completed by the candidate

Registration No: 1025306

Candidate: Abimbola Efunogbon

Degree of: PhD

Research Institute: IRAC

Research Topic: Optimal 5G Network Sub Slicing Orchestration in a Fully Virtualised Smart Company Using Machine Learning (ONSSO)

External Funding: None

The candidate is required to summarise in the box below the ethical issues involved in the research proposal and how they will be addressed. In any proposal involving human participants the following should be provided:

- clear explanation of how informed consent will be obtained,
- how will confidentiality and anonymity be observed,
- how will the nature of the research, its purpose and the means of dissemination of the outcomes be communicated to participants,
- how personal data will be stored and secured
- if participants are being placed under any form of stress (physical or mental) identify what steps are being taken to minimise risk

If protocols are being used that have already received University Research Ethics Committee (UREC) ethical approval then please specify. Roles of any collaborating institutions should be clearly identified. Reference should be made to the appropriate professional body code of practice.

This research will be conducted using 5G dataset which is available on the public domain or from 5G test beds.

The 5G dataset and process for acquiring it does not cause any ethical issues.

Answer the following question by deleting as appropriate:

1. Does the study involve vulnerable participants or those unable to give informed consent (e.g. children, people with learning disabilities, your own students)?
   No

   If **YES**: Have/will Researchers be DBS checked?
   Yes **No**

2. Will the study require permission of a gatekeeper for access to participants (e.g. schools, self-help groups, residential homes)?
   No

3. Will it be necessary for participants to be involved without consent (e.g. covert observation in non-public places)?
   No

4. Will the study involve sensitive topics (e.g. sexual activity, substance abuse)?
   No

5. Will blood or tissue samples be taken from participants?
   No

6. Will the research involve intrusive interventions (e.g. drugs, hypnosis, physical exercise)?
   No

7. Will financial or other inducements be offered to participants (except reasonable expenses)?
   No

8. Will the research investigate any aspect of illegal activity?
   No

9. Will participants be stressed beyond what is normal for them?

October 2014

No

10. Will the study involve participants from the NHS (e.g. patients) or participants who fall under the requirements of the Mental Capacity Act 2005?

* No

If you have answered yes to any of the above questions or if you consider that there are other significant ethical issues then details should be included in your summary above. If you have answered yes to Question 1 then a clear justification for the importance of the research must be provided.

*Please note if the answer to Question 10 is yes then the proposal should be submitted through **NHS research ethics approval procedures** to the appropriate **NRES**. The UREC should be informed of the outcome.

Checklist of documents which should be included:

Project proposal (with details of methodology) & source of funding	x
Documentation seeking informed consent (if appropriate)	
Information sheet for participants (if appropriate)	
Questionnaire (if appropriate)	

(Tick as appropriate)

Applicant declaration

I understand that I cannot collect any data until the application referred to in this form has been approved by all relevant parties. I agree to carry out the research in the manner specified and comply with the statement of ethical requirements on page 1 of this form. If I make any changes to the approved method I will seek further ethical approval for any changes.

Signature of Applicant: ..... Date: .....06/07/2021.....

Signature of Director of Studies: ..... Date: 06/07/21.....

This form together with a copy of the research proposal should be submitted to the Research Institute Director for consideration by the Research Institute Ethics Committee/Panel

Note you cannot commence collection of research data until this form has been approved

SECTION B To be completed by the Research Institute Ethics Committee:

Comments:

Approved

Signature Chair of Research Institute Ethics Committee:



Date: 7/7/2021

This form should then be filed on the student's record

If in the judgement of the committee there are significant ethical issues for which there is not agreed practice then further ethical consideration is required before approval can be given and the proposal with the committees comments should be forwarded to the secretary of the UREC for consideration.

There are significant ethical issues which require further guidance

Signature Chair of Research Institute Ethics Committee:

Date:

This form together with the recommendation and a copy of the research proposal should then be submitted to the University Research Ethics Committee

Appendix B: Research Data Management Plan

Overview

Project title
Optimised 5G Network Sub Slicing Orchestration in a Fully Virtualised Smart Company Using Machine Learning (ONSSO)
Student name and Research Institute
Abimbola Efunogbon - Institute for Research in Applicable Computing (IRAC)
Supervisor(s)
Note: the Director of Studies is the Data Steward
**Dr. Enjie Liu* *Dr. Renxi Qiu*
Project description: aim, objectives, main research methods used (2-3 sentences will suffice)
This research aims to develop a framework for efficiently orchestrating 5G network slices using generative AI. and machine learning by sub slicing the slices. The research involved analysing existing datasets from Kaggle and synthetic data generation to evaluate the performance of the proposed model which was then validated by using python and MATLAB simulations. Key methods involve machine learning techniques, including GANs, using PyTorch.

Compliance

When you submit your DMP you are confirming that you have read and understood all of the legislative, policy and contractual requirements that apply to your project.

University policy requirements
Compliance with the University of Bedfordshire's policies on data management, Research Ethics, and Intellectual Property rights.
University policy or guidance
University of Bedfordshire GDPR Guidance for Researchers (BREO sign-in required)
University of Bedfordshire Research Ethics Policy
University of Bedfordshire Internet-Mediated Research Guidelines (BREO sign-in required)
University of Bedfordshire Intellectual Property Policy
University of Bedfordshire Compliance with the Concordat to Support Research Integrity
Legal requirements
UK Data Protection Act (2018)
UK Legislation or framework

Contractual requirements	
Not Applicable	
Name of funder	**Data policy URL**

Gathering data – see 'Assessment of Existing Data' and 'Information on New Data' in the RDM Checklist. Cross-reference with your Data Protection Impact Assessment.

Description of the data
3.1.1 Types of data – existing and/or new
• *Existing secondary data: Kaggle datasets on 5G network slicing* • *New primary data: Synthetic data generated for model testing* • *Software code: Scripts developed for data analysis and model implementation*
3.1.2 Format and scale of the data
• *Excel files for data analysis* • *Python scripts and Google Colab notebooks for code implementation* • *CSV files for long-term archiving*
Data collection methods
• *Data will be collected from Kaggle datasets, specifically from the Dataset Unicauca Google Drive. Additional synthetic data will be generated using custom algorithms.*
Development of original software
• *Scripts will be developed using Python, utilising PyTorch libraries and. Dependency management will be handled using requirements.txt files. The software will be primarily for this project, but potential future versions may be released.*

Working with data – see 'Quality Assurance of Data' and 'Backup and Security of Data' in the RDM Checklist. Cross-reference with your Data Protection Impact Assessment.

Short- and medium-term data storage arrangements
Data will be stored on Google Drive for collaboration and backup. Weekly automatic backups will be scheduled. Local backups will also be maintained on an external hard drive
Control of access to data and sharing with collaborators (beyond your supervisory team)
Access will be granted to the supervisory team and collaborators via Google Drive. Data transfers to other institutions will be secured through encrypted email.
File organisation and version control
Files will be organised by project phase (e.g., Data Collection, Analysis). Naming conventions will follow the format: "DatasetName_Version_Date". Version control will be managed using Git.

Documentation that will accompany the data
Documentation will include a data dictionary and README files, detailing the dataset structure and usage instructions. These will be stored alongside the data files in Google Drive.

Archiving data – see 'Preparation of Data for Sharing and Archiving' in the RDM Checklist

Selection of data to be retained and deleted at the end of the project
Data used for publications will be retained, while redundant data will be deleted securely. Personal data will be anonymized or destroyed according to data protection policies.
Data preservation strategy and retention period
Data will be archived on Google Drive and an external hard drive for a minimum of five years following project completion.
Maintenance of original software
The original code will be archived in a GitHub repository. Future development may continue, with access rights managed by the original authors.

Sharing data – see 'Preparation of Data for Sharing and Archiving' in the RDM Checklist

Justification for any restrictions on data sharing
Data will be shared openly unless sensitive information is involved. Restrictions will be applied for any proprietary or personal data.
Arrangements for data sharing
Data will be shared via GitHub and Google Drive. Repositories will be maintained for public access after project completion.

Implementation – see 'Responsibilities' in the RDM Checklist

Review of the Data Management Plan
The DMP will be reviewed quarterly, with key reviews scheduled before and after significant project milestones The author will be responsible for updates.
Special resources required for the project
Additional cloud storage may be required if data exceeds local capacity.
Further training needs
No.

Appendix C: Source Code

https://github.com/abimbola1/Optimal_5G_Slice_Orhestration