Towards Automating IP-Network Operations with Machine Learning from Raw Network Data

by

Ayse Rumeysa Mohammed

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy Degree
in
Electrical and Computer Engineering
School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

"Memory is attention through time."

Alex Graves, 2020.

Abstract

The ever-increasing size and complexity of communication networks today complicate Network Operation Centers (NOC) to function efficiently in manually operated tasks such as network status detection, network fault localization, cost-aware traffic engineering, failure management, and network quality assurance. These tasks have traditionally been managed by expert technicians who make decisions on when and where to take which actions based on specific network rules. Due to the complexity of the process, NOC actions are still performed manually. However, automating this process could be a valuable input for network providers and service operators.

In this context, we developed an Artificial Intelligence based (AI-based) action recommendation engine (ARE) which, as its name suggests, recommends the best available operational expenditure aware (OPEX-aware) action, either with (Stateful ARE) or without (Stateless ARE) measuring the network state. Our experimental results show that Stateful ARE can recommend the suitable action and yield up to 99% accuracy. This high accuracy percentage is due to the correct classification of the *Normal* state, which represents 64.5% of the dataset, and its corresponding action of *Do Nothing*, which accounts for 68.3% of all actions.

While Stateful ARE's overall accuracy is satisfactory, it was unable to achieve this performance in minority classes, and it suffered from performance degradation due to state classification process. Therefore, we introduced Stateless ARE, which recommends actions without measuring the network state. The initial results of Stateless ARE using a Feed Forward Neural Network (FFNN) did not exceed Stateful ARE's performance. The classification accuracy of minority classes were still around 89% and 93%, but it outperformed the static network, indicating that it could be improved with further optimization techniques.

Based on this insight, we introduced state-of-the-art Transformer model as Stateless ARE model. The transformer model significantly improved the accuracy of the minority classes to 97% and 99%, which other methodologies struggled to classify. This result shows that the transformer model can be an effective tool in improving the performance of action recommendation engines.

Acknowledgements

I would like to convey my heartfelt appreciation to my thesis advisor, Prof. Shervin Shirmohammadi, for his invaluable guidance and unwavering support throughout my thesis work. He was always approachable whenever I encountered any obstacles or felt stuck in my research.

I would like to extend my sincere gratitude to Prof. Daniel Amyot for his unwavering support during the unprecedented times of the COVID-19 pandemic. His mentorship has surpassed the conventional teacher-student relationship and has been a guiding light throughout the arduous journey of my PhD. I am deeply indebted to him for his empathy, guidance, and support which have significantly contributed to my academic success, especially during the darkest moments of modern history.

I would also like to acknowledge Hussein Al Osman for his valuable insights. His profound understanding and expertise in AI have been instrumental in shaping my research work. I am truly grateful for his willingness to share his knowledge.

I would like to express my heartfelt appreciation to my lovely husband, Shady Mohammed, for his continuous support and encouragement throughout my academic journey. His constant presence and sharp intelligence have been a source of motivation, enabling me to achieve excellence in my work. His patience, enthusiasm, and motivation have been the pillars of my success, and I am truly grateful for his commitment and belief in me. I cannot thank him enough for holding my hand and helping me navigate through the ups and downs of my research journey.

I would like to express my deepest appreciation to my mother, whose unwavering support and selfless sacrifices have played an immeasurable role in my academic success. Throughout her entire life, she has worked tirelessly to provide me with everything I needed to thrive and achieve my goals. Her encouragement and dedication to my education have been a constant source of motivation, even during the toughest times. I am honored to be the first and only person in my family to obtain a PhD degree, and it is all thanks to her unrelenting love and devotion. I am forever grateful for her sacrifices, and I will always cherish the memories of her unwavering support. Thank you, Mom.

My sincere appreciativeness goes to Salma Al-Sawy - *my life long friend* and Namrata Bagaria - *my companion.*

Table of Contents

List of Tables

List of Figures

Abbreviations

AI Artificial Intelligence

API Application Programming Interface

ARE Action Recommendation Engine

AS Autonomous System

AUC Area Under Curve

CNN Convolutional Neural Network

Cong Congestion

CSP Communication Service Providers

DASH Dynamic Adaptive Streaming over HTTP

DL Deep Learning

DNN Deep Neural Networks

DT Decision Trees

EOS End-of-sequence

FFNN Feed Forward Neural Network

FN False Negative

FP False Positive

FPR False Positive Rate

FTP File Transfer Protocol

GB Gradient Boosting

GNN Graph Neural Network

GNS3 Graphical Network Simulator-3

GRU Gated Recurrent Unit Networks

IGP Interior Gateway Protocol

IOS Internetworking Operating System

IP Internet Protocol

ISP Internet Service Provider

IT Information Technology

KNN K-Nearest Neighbour

LSTM Long Short-term Memory Networks

ML Machine Learning

MLOps Machine Learning Operations

MPLS Multiprotocol Label Switching

MTSIT Multivariate Time-Series Imputation with Transformers

NLP Natural Language Processing

NN Neural Networks

NOC Network Operation Centers

NPI Network Physical Issue

OPEX Operational Expenditures

OS Operating System

OSPF Open Shortest Path First

QoE Quality of Experience

QoS Quality of Service

ReLU Rectified Linear Activation Unit

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic

RTT Real-Time Text

SDN Software Defined Networks

SLA Service Level Agreements

SOS Start-of-sequence

SVM Support Vector Machine

TCP Transmission Control Protocol

TE Traffic Engineering

TFT Temporal Fusion Transformers

TN True Negative

TP True Positive

TPR True Positive Rate

US United States

XGB Extreme Gradient Boosting

Chapter 1

Introduction

1.1 Motivation

The internet has become one of the rapidly growing technologies in the world, with almost 5 billion active users reported in June 2022 [6]. This growth and expansion has led to an increase in various ways of communication, as reported in [44], where 9 billion clients receive services from Communication Service Providers (CSP) for remote transmission of voice, data, texts, sound and images over networks. CSPs are responsible for providing high-quality services to their clients in accordance with Service Level Agreements (SLA) while also keeping their operational expenditures (OPEX) to a minimum. To maintain a healthy network with high quality operations and reduced costs, network experts and IT administrators operate at Network Operation Centres (NOCs) [13], see Figure 1.1. At NOCs, network professionals supervise the detailed status of the network by measuring and tracking it.

The responsibilities of network experts at NOCs are vast and include troubleshooting, problem identification, localization, isolation, and diagnosis, network monitoring and updates, performance and quality reporting, quality control and assurance, policy enforcement, threat analysis, and more. However, traditionally, such network supervision has been done manually. This approach is currently inefficient and impractical, and it becomes increasingly insufficient with the expansion of communication networks. In the near future, it will become obsolete due to the exponential increase in hardship introduced by the expansion. With the current service conditions, CSPs promise to keep uptime at 99%+, but this promise is becoming more challenging, time-consuming, and, therefore, more expensive. According to Uptime Institute's 2022 Outage Analysis Report, over 60% of network

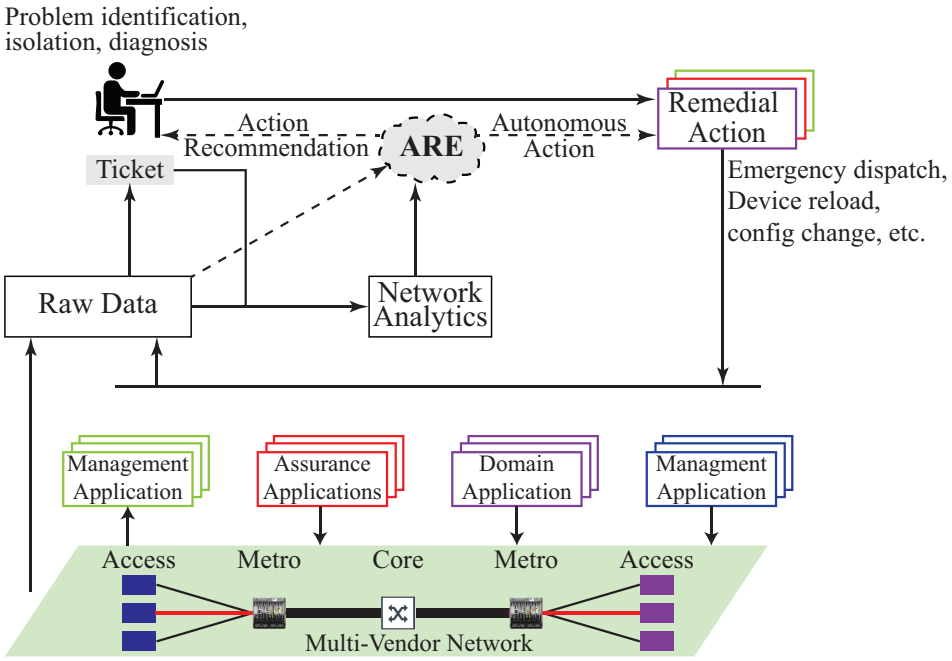


Figure 1.1: A typical NOC. The ARE component is proposed by this work.

outages cost more than $100,000 and 15% of them cost more than $1 million to the businesses [7]. Therefore, automating the manual maintenance process helps to reduce the cost of network outages.

However, past attempts to improve these automations have faced challenges due to the large amount of data and the lack of clear rules to solve these complex problems, even with well-trained personnel. Therefore, we need modern methods to address these issues and automate processes to adapt quickly to the increasing data. This is where Artificial Intelligence (AI) comes in. AI has several advantages over manual operations and rule-based systems in this context. When faced with new problems, AI can quickly find network status and suggest actions. AI can also adapt to different situations. Furthermore, when people are under pressure, they tend to make more mistakes, while AI performs well. AI is faster, can handle errors, and can scale when using the right models. By automating tasks like network status detection, fault identification, efficient traffic management, failure handling, and network quality assurance, which were previously managed by expert technicians following specific network rules, network providers and service operators can significantly reduce the time and cost of network maintenance while ensuring high-quality services for their customers.

1.2 Research Problem

Our industrial partner Ciena formulated the challenges faced by network operators and IT administrators at NOCs. With their guidance, we built an emulator on GNS3 to tackle these challenges. Specifically, we focused on automating network operations to reduce costs and improve network performance. Through our side by side experiments with Ciena, we demonstrated that AI-based systems can help automate two critical network operations: failure management and OPEX-aware traffic engineering. To achieve this, we developed an AI-based Action Recommendation Engine and coined the term *ARE*, which is a constantly running decision-making system designed to recommend suitable actions to fix network failures and engineer traffic to keep OPEX at a minimum. ARE has two variations: Stateful ARE and Stateless ARE. Stateful ARE takes the network state as an input and recommends suitable actions based on that state. In contrast, Stateless ARE recommends the best action without network state measurement.

To the best of our knowledge, our work is the first to propose a system that chooses the best action to remedy network failures autonomously. In other words, ARE can take actions without any human intervention, which is an important step towards more automated and efficient network management. We conducted experiments to test the efficacy of ARE and found that its results were very close to those of NOC technicians, indicating its potential to reduce the workload on human operators while maintaining network performance.

During our experiments, we also found that eliminating network state detection could optimize ARE further. This led us to develop the stateless variation of ARE, which introduced even more automation and speedy operations. While the initial results of Stateless ARE did not outperform its Stateful equivalent, it did outperform a static model, indicating its potential for our research problem.

We further advanced stateless ARE by applying transformers to recommend actions from raw data. Transformers are a type of neural network architecture that has achieved state-of-the-art performance in various natural language processing (NLP) tasks. Our experiments showed that applying transformers to ARE significantly improved the accuracy of minority classes, which are instances that occur less frequently in the dataset. The improvement demonstrated the great potential of our approach to bring about further improvements in network operations management.

Our work is crucial because it addresses the challenges faced by network operators and IT administrators at NOCs. The current manual approach to network management is inefficient and impractical, and with the expansion of communication networks, it will become obsolete. The promise of CSPs to maintain uptime at 99%+ is becoming more

challenging and expensive. The automation of network maintenance processes can reduce costs and improve network performance while ensuring high-quality services to clients. By introducing AI-based systems like ARE, we can automate tasks such as detecting network status, fault localization, cost-aware traffic engineering, failure management, and network quality assurance, which are currently managed by expert technicians who make decisions based on specific network rules.

In conclusion, our work demonstrates the potential of AI-based systems to automate critical network operations and improve network performance while reducing costs. ARE can autonomously recommend suitable actions to fix network failures and engineer traffic to keep OPEX at a minimum. Our experiments show that ARE's results are comparable to those of NOC technicians, indicating its potential to reduce the workload on human operators while maintaining network performance. The development of stateless ARE and its optimization with transformers opens up new possibilities for further improving network operations management.

1.3 Contributions

The increasing complexity and heterogeneity of modern communication networks pose significant challenges for network operators to manage and maintain. With the exponential growth in the number of connected devices and the demand for high-speed internet, network failures are inevitable, and the detection and recovery process can be both time-consuming and expensive. AI-based systems provide an efficient and effective solution to this problem by automating the detection and recovery process in near real-time. Hence, we designed two AI-based systems that aim to automate the process of restoring a network to its normal state in near real-time. The first system is a Stateful AI-based system that predicts the network state, recommends the best suitable action to restore the network to its normal state, and operates in near real-time. The second system is a stateless AI-based system that recommends the best suitable action to restore the network to its normal state without predicting the network state and operates in near real-time.

The Stateful AI-based system operates in a closed-loop system that receives QoS and QoE metrics as inputs and predicts the network state. Based on the predicted network state, the system recommends the best suitable action to restore the network to its normal state. The system consists of two cascaded ML models, where the first model predicts the network state and the second model recommends the best suitable action. The system is trained with a large dataset of QoS and QoE metrics and has achieved high accuracy in predicting the network state and recommending the best suitable action. This is the first

system we designed. It is an important contribution because predicting the network state accurately is a challenging task that requires sophisticated algorithms and techniques. Our system achieves this by using state-of-the-art ML models to analyze QoS and QoE metrics collected from the network. The system's ability to accurately predict the network state is critical to selecting the best action that restores the network to its normal state.

The stateless AI-based system, on the other hand, eliminates the network state detection process and only recommends the best suitable action to restore the network to its normal state. The system receives QoS and QoE metrics as inputs and recommends the best suitable action to restore the network to its normal state. The system is trained with a large dataset of QoS and QoE metrics and has achieved high accuracy in recommending the best suitable action. This is the second system we designed. Eliminates the need for predicting the network state is a significant contribution because it simplifies the system design and eliminates the risk of error propagation caused by misclassifying the network state. Furthermore, this system accelerates the automation process by removing the state detection process, which can be time-consuming and computationally expensive.

Our contributions lie in designing and implementing two AI-based systems that operate in near real-time and automate the process of restoring a network to its normal state. The stateful AI-based system uses a closed-loop system that predicts the network state and recommends the best suitable action. The stateless AI-based system eliminates the network state detection process and only recommends the best suitable action. Both systems have achieved high accuracy in predicting or recommending the best suitable action, respectively. These systems can help network operators to restore the network to its normal state in near real-time and improve network availability and reliability.

The creation of this closed-loop system is a significant achievement because it allows for the automatic delivery of predicted actions on the network. With this system, network administrators no longer have to manually diagnose network faults, which can be time-consuming and prone to errors. The closed-loop system can diagnose network faults automatically and recommend the best available action, which can save network administrators a lot of time and resources. Furthermore, with the development of Stateless ARE, the performance degradation of Stateful ARE caused by misclassification errors can be solved viably since it eliminates the need for a network state detector, which simplifies the model and reduces the computational complexity and training time. Moreover, it is scalable, easier to monitor and maintain, and can operate in near real-time, making it suitable for critical network applications.

Hence, we can summarize the contributions as follows:

- Design and implementation of an AI-powered network state classification system that

automates the restoration process of a network in near real-time at NOCs.

- Design and implementation of a closed-loop system enabling automatic execution of predicted actions on the network.
- Design and implementation of stateless action recommendation system, reducing computation time and memory requirements compared to its stateful counterpart.

1.4 Publications & Outputs

- **Journal.** Shady A Mohammed, Ayşe Rumeysa Mohammed, David Côté, and Shervin Shirmohammadi. "A machine-learning-based action recommender for network operation centers." IEEE Transactions on Network and Service Management, 18(3):2702–2713, 2021. See [51].
- **Journal.** Ayşe Rumeysa Mohammed, Shady A Mohammed, David Côté, and Shervin Shirmohammadi. "Machine learning-based network status detection and fault localization." IEEE Transactions on Instrumentation and Measurement, 70:1-10, 2021. See [48].
- **Conference.** Ayşe Rumeysa Mohammed, Shady A Mohammed and David Côté, and Shervin Shirmohammadi. "Stateless ARE: Action Recommendation Engine without Network State Measurement." IEEE International Symposium on Measurements & Networking, 1:6, 2022. See [49].
- **Conference.** Ayşe Rumeysa Mohammed, Shady A Mohammed, and Shervin Shirmohammadi. "Machine learning and deep learning based traffic classification and prediction in software defined networking." IEEE International Symposium on Measurements & Networking, 1:6, 2019. See [50].
- **Patent.** David Côté, Ayşe Rumeysa Mohammed, Shady A. Mohammed, and Shervin Shirmohammadi. "Action Recommendation Engine based on Machine Learning to support closed-loop applications for telecommunications networks", Attorney Docket Number: 10.2767, Patent Grant Number: 11316752.
- **Award.** "Action Recommendation Engine based on Machine Learning to support closed-loop applications for telecommunications networks" – Best idea in 2020 Ciena's Think Tank Forum.

- **Industrialization.** The technology developed in this work is being commercialized by the University of Ottawa and our industrial partner Ciena Corp. under the Idea to Innovation Natural Sciences and Engineering Research Council of Canada Program (I2I NSERC).

1.5 Outline

The remainder of this thesis is structured into six chapters, each covering important aspects of the research conducted in this work.

Chapter 2 focuses on providing a detailed explanation of ML and DL models implemented during the experiments. This chapter discusses the key concepts, algorithms, and architectures used in the models, the rationale behind selecting each model along with their strengths and limitations.

In Chapter 3, we present a comprehensive review of the related work in the field of network operations and automation. This chapter covers a wide range of literature, including research on ML-based decision making systems, OPEX-aware traffic engineering, network failure management, and many more.

Chapter 4 introduces the developed system, its architecture, and the testbed used in this work. We provide a detailed description of the system components, their functionalities, and the system architecture. We also discuss the testbed setup, the datasets used in the experiments along with how they are collecting and augmented, and the evaluation metrics.

Chapter 5 evaluates the performance of the algorithms implemented in this work and analyzes the results. This chapter discusses the effectiveness of the ML and DL models, their ability to make accurate recommendations to ou research problem, and the impact of different parameters on the system's performance.

Chapter 6 concludes our work and discusses future research venues. This chapter summarizes the key findings of the study, highlights the contributions made in this work, and extensively discusses potential areas for future research. We also provide recommendations for practitioners and researchers who are interested in further exploring the use of AI in network operations and automation, specifically applying trasnformers into time series problems related to computer networks and traffic engineering.

Chapter 2

Background

2.1 Machine Learning

Machine Learning (ML) is the science of building algorithms to solve a possible problem by creating a statistical model based on a dataset [14]. ML could possibly have more than 10 types of learning; supervised, ensemble, unsupervised, reinforcement, semi-supervised, transfer, and etc. In this thesis we will focus the first two types of learning.

2.1.1 Machine Learning Types

2.1.1.1 Supervised Learning

Supervised learning is the approach where the input data come with their associated labels. the input data includes descriptive features and the label, usually corresponding the class that the data belongs to. Since the training set is composed of inputs with their *correct* outputs, the model learns the right values. As input data is introduced to the model to train it, it adjusts its weights to fit the inputs by measuring via loss function which aims to minimize the error.

Supervised learning is generally used with classification and regression problems. Classification is the arrangement of the data into categories based on their shared characteristics while regression is the predictive modelling where it takes independent variables as inputs and projects dependent variables as outputs. The most common supervised learning algorithms are neural networks (NN), decision trees (DT), and etc. Some examples of real life

application of supervised learning models can be stated as (i) image and object detection such as text categorization, face detection, signature recognition (ii) sentiment analysis such as social media monitoring, employee feedback tracking (iii) predictive analytics such as stock market analysis or housing price predictions (iv) spam and fraud detection.

Although supervised learning is implemented for numerous machine learning problems, certain challenges are faced during implementation. The most devastating mistake is working with a dataset having mislabels due to human error or irrelevant input feature. These mistakes causes the model to be trained with inaccurate labels or completely wrong data. When dataset is large such mistakes are often compensated. However when the data is scarce, its effects are more detrimental. In general, they are caught during data exploration phase which is the first step in understanding the input data through data visualization and statistical techniques. Another challenge is that training for supervised learning is computationally expensive. Therefore it introduces an extra challenge to deal with big data. Since supervised learning is trained with labels, it cannot cluster data by figuring out the features on its own. Furthermore, in order to achieve optimal outcomes, it is imperative that each class is endowed with a sufficient quantity of precisely labeled input data, thereby enabling the model to effectively distinguish between distinct classes.

2.1.1.2 Ensemble Learning

Ensemble learning is the approach to enhance the predictive performance by combining several learning algorithms usually by using the same base learner. However, when diverse models are used, ensemble learning is highly likely to produce high performance. Ensemble learning is generally used with classification and prediction problems. The most common ensemble learning algorithms are DT, GB, XGB, and etc.

There are three main classes of ensemble learning; bagging, boosting, and stacking. Bagging fits multiple decision trees on different data points in the same dataset. Then it averages their predictions. Boosting adds the ensemble models successively to fix the mistake of the previous models. Then it takes the weighted average of the predictions. Stacking fits multiple models on the same data. Then it utilizes another model to decide the optimum combination of the predictions.

Ensemble learning is popularly implemented in wide range of applications. Its popularity is due to its simplicity of implementation and success on various predictive modeling problems. Some applications can be listed as (i) facial recognition in as ID or emotion identification (ii) speech recognition (iii) computer vision as in detection change in a landscape, i.e., deforestation monitoring (iv) fraud detection as in credit card frauds.

Despite their popularity in implementation, the fundamental issues revolving around ensemble learning are challenging to address. Ensembling is more meagerly interpretable because its output is not explainable. Since the output is hard to understand, any selection regarding the model parameters such as ensemble size or model selection in ensembling, is not intuitive [76]. Therefore any wrong selection decreases the predictive accuracy significantly more than an individual model.

2.1.2 Machine Learning Models

2.1.2.1 Decision Tree (DT)

DT is an acyclic tree shaped graph that makes decision through recursive binary splitting [62]. In every tree branch, a feature from the input data is examined. A decision regarding this feature is made according to the threshold value, see Figure 2.1.

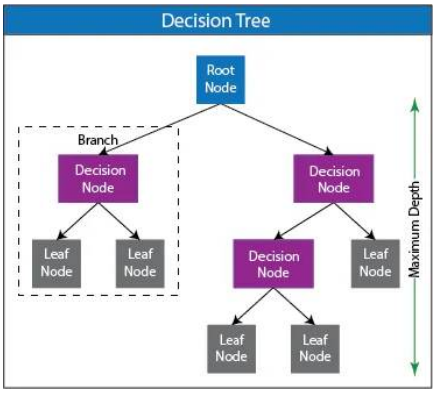


Figure 2.1: Decision Tree

DT is easy to visualize and therefore, interpret. Due to its nature, it executes feature selection. It also doesn't require a detailed data preparation and data preprocessing. However, it is not a very stable model, for example; its variance is high. Since the tree graph is being generated along the training, even the slightest variations in the input data could bring about a different tree. This problem is usually mitigated by ensemble learning classes such as bagging or boosting. In addition, if the model parameters are not managed well, it could create over complex models which results in overfitting. It also cannot handle well imbalanced datasets, for instance; when one class in the dataset dominates the other classes, DT tends to create biased trees to increase its performance score.

2.1.2.2 Gradient Boosting (GB)

As explained in Section 2.1.1.2, boosting converts weak learners into strong learners by modifying the initial data set. Hence, we can define that GB is a predictive model which ensembles weak prediction models [21]. Then it tests them on the dataset, identifies their shortcomings, repeats this process till the final ensemble model is generated using a gradient descent optimized loss function that gives the minimum error, see Figure 2.2. Traditionally, the weak base model of GB is chosen to be DT. Number of trees, learning rate, and depth of trees are three parameters that affect GB's accuracy.

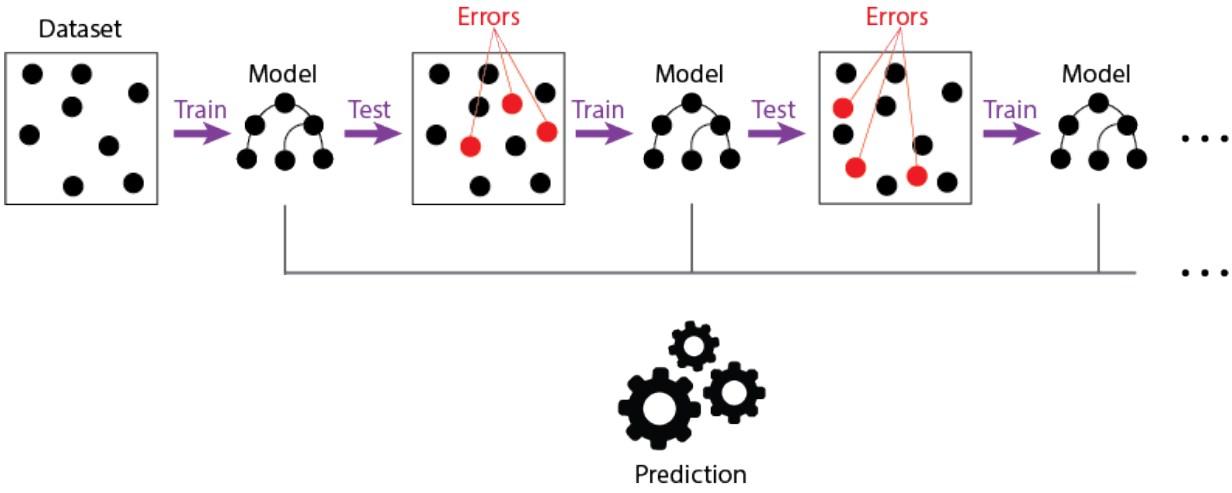


Figure 2.2: Gradient Boosting

While combining weak learners increases the accuracy, it decreases the model's interpretability [79] because the model becomes too complex and understanding the decisions done by hundreds of trees is not intuitive. When the model becomes too large, it is prone to overfit. Additionally, outliers substantially distort the model since their residual is larger than non-outlier data points.

2.1.2.3 Extreme Gradient Boosting (XGB)

XGB is built on GB framework. It includes decision trees as its base estimators just as GB, see Fig2.3. XGB uses a more regularized formalization that overcomes over-fitting and results in better accuracy [16]. This unique ability makes the algorithm to run faster and increase scalability. However it doesn't perform well on dispersed or sparse data.

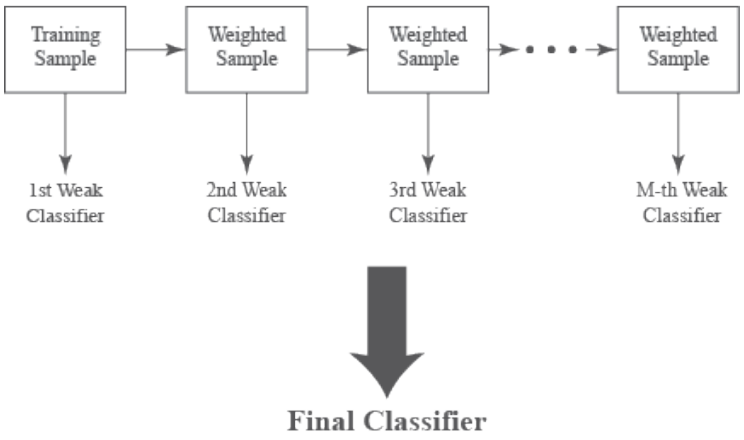


Figure 2.3: eXtreme Gradient Boosting

2.2 Deep Learning

Deep Learning is an advanced framework for supervised learning. It adds extra layers and withing these layers, it adds extra units to increase the model's capacity to solve complex problems often with multiple dimensions [24].

2.2.1 Deep Learning Models

2.2.1.1 Feed Forward Neural Networks (FFNN)

Feed Forward Neural Networks (FFNN), are neural networks where information travels through forward direction, i.e., the network doesn't have a feedback system where the output is fed back into the framework. FFNN maps the inputs into a category and through iterations, learns the best approximation values for the model parameters.

FFNN consist of multiple layers of networks. The number of layers define the *depth* of the network. The first layer of the network corresponds to the input layer while the last layer is the output layer, and the other layers are named as hidden layers. The dimensionality (the number of neurons) of the hidden layers defines the *width* of the model, see Figure 2.4.

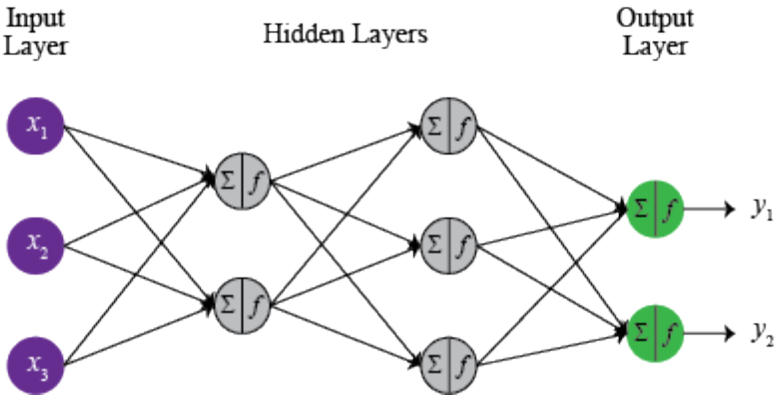


Figure 2.4: Feed Forward Neural Networks

2.2.1.2 Graph Neural Network (GNN)

A graph neural network is a framework that represents DNNs on graph data [26]. A graph is a structure with complex relationships and interdependencies between its objects.

Graphs are data structures with nodes and edges. They can be directed, undirected, cyclic, and acyclic, see Figure 2.5, 2.6, and 2.7. In directed graphs, edges have directions while in undirected, edges do not specify the direction. In addition, if a graph has at least one cycle where the nodes are connected in a cyclic shape, then the graph is called cyclic.

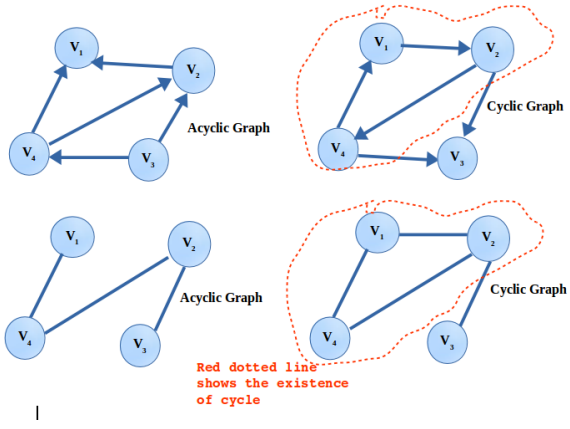


Figure 2.5: Cyclic Graph

Many data can naturally be represented as graphs, for example, transportation net-

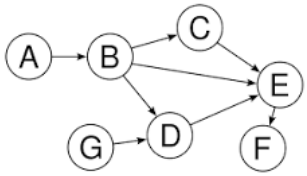


Figure 6 : Acyclic Graph

Figure 2.6: Acyclic Graph

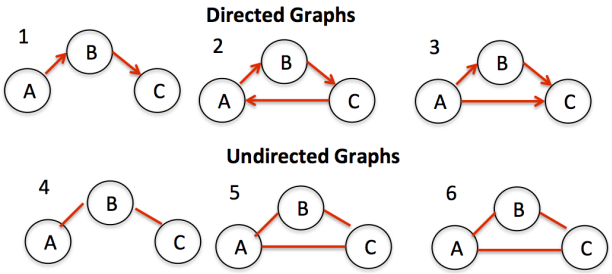


Figure 2.7: Directed and Undirected Graph

works. The edges are the roads and the nodes are the intersections of the roads. The road length, current vehicle speed, and historical speed could be node features. By identifying these characteristics, we can easily extract information such as estimated time of arrival of a vehicle, from the graph representation.

In other words, nodes represent objects, and edges represent relationship between these objects. To this end, we define *state* that contains the information about a specific node in its own neighborhood, i.e., state is the concept that generates the output in a GNN. Hence, the complexity in building a GNN increases linearly with number of nodes, the number of edges, the number of hidden units in the NN but it increases exponentially with the state dimension [66].

Analyzing graph data is challenging for current ML methods because they are build to work on sequential (such as text) or same structured (such as images) data. However, graph data, generally, does not have a fixed form with fluctuating shape of unordered nodes. Also the instances within graph data is associated to its neighbor with various links. This concept is called *node embedding*. It means that instances are embedded into a d-dimensional space based on their similarities to each other, i.e., closer points in the data are similar to each other. These points are embedded using the encoder function whose inputs are *locality*, *aggregate information*, and *stacking multiple layers*. Locality

information is captured by computational graph. The node n is connected to its neighbors and its neighbors' neighbors, see Figure 2.8. Locality sets insights to graph structure and retrieves feature information. When the locality is build, aggregation is initiated. Aggregation is an order- and permutation-invariant process where the input feature vectors are aggregated to create a new feature vector at each node which then passed on to the next layers until it reaches to the target node 2.9. This aggregation is applied via forward propagation rule.

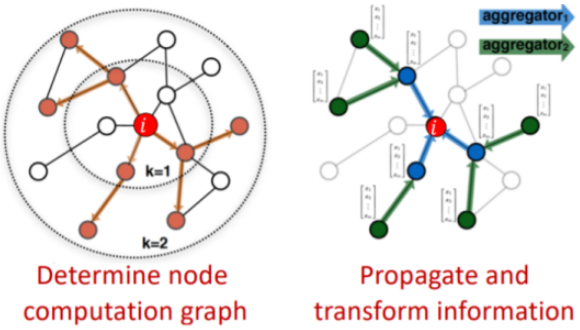


Figure 2.8: Locality Information

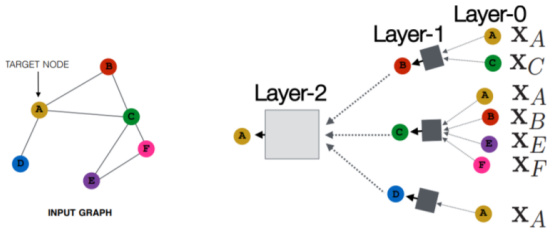


Figure 2.9: Aggregation

GNNs' design is considerably different than any other NNs. One prominent difference is that they typically have two or three layers whereas other NN models could have hundreds of layers. The other one addresses the sparsity. GNNs work well with sparse data while other NNs suffers from it since when the data is sparse, the network learns mostly zeros which deteriorates the performance. Hence, GNNs architecture differs noticeably, see Figure 2.10. GNNs take the graph representation of data as an input. Throughout the GNN layers, it applies computational modules (such as aggregation, sampling, pooling, etc.) which outputs node embeddings, i.e., the feature and topological information of instances with their relations to each other. Then, the loss function is designed. It allows us

to use all learning types (supervised, semi-supervised, unsupervised, and graph learning) and all level tasks (node, edge, and graph levels).

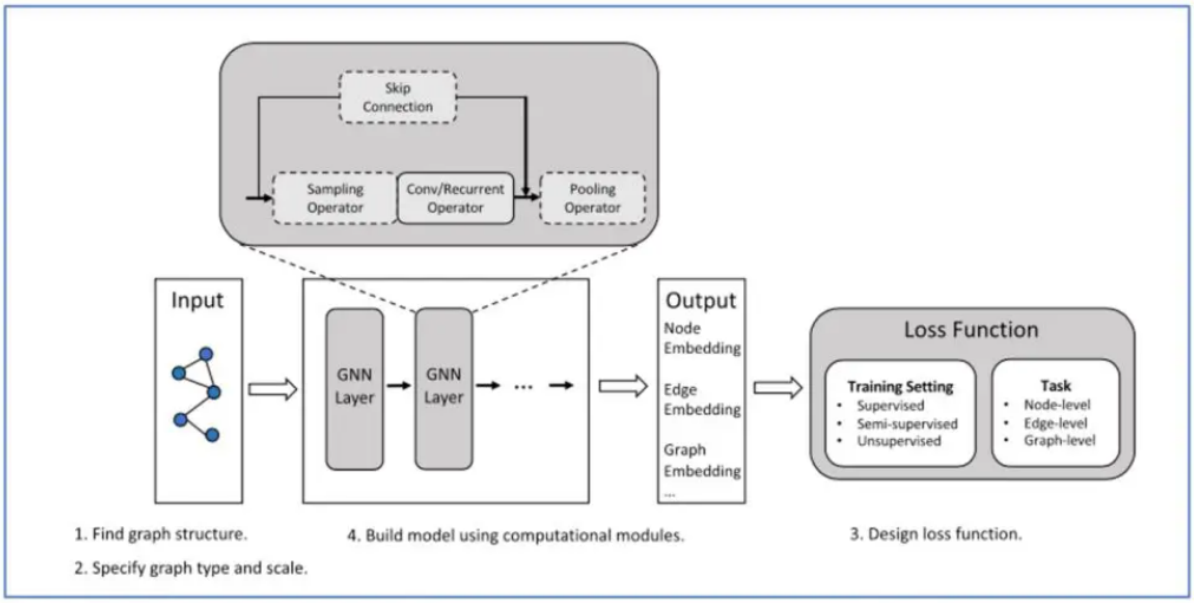


Figure 2.10: Design Pipeline for GNN

GNNs can be applied in various fields including but not limited to image classification [61], object detection [77], semantic segmentation [60], physics [70], relation extraction [86], and web page ranking [67].

2.2.1.3 Transformers

The Transformer is a DL architecture that was initially designed specifically for natural language processing (NLP) tasks such as language translation, text summarization, speech recognition, and language modeling [74]. Contrary to current NLP models, which rely on recurrent neural networks (RNNs) and convolutional neural networks (CNNs), transformer utilizes a self-attention mechanism to process data sequences. This allows the model to access different parts of the input sequence regardless of distance when making predictions, instead of processing the input sequentially like an RNN. Giving weight to important parts of the data makes transformers grasp the recurring patterns with long-term dependencies.

Transformer models are essentially composed of encoder and a decoder. The encoder takes in the input sequence and generates several sequences of hidden states. Then, the

decoder takes in these hidden states and generates the output sequence. The key feature in this process is the multi-head attention which enables the model to attend to different parts of the input sequence in parallel. Hence, the model is able to grasp both local and global feature dependencies of the input sequential data and increase the model's ability to learn representations at different levels of abstraction.

One of the most devastating issues that causes model performance degradation is vanishing gradient problem. Transformer overcomes this common problem and stabilizes itself by using (i) residual connections which are skip connections allowing information to move from one layer to another and (ii) layer normalization.

Transformer architecture can be broken down into three main components: the input embeddings, the encoder, and the decoder, see Figure 2.11. Firstly, the input sentence is split into single words using tokenizers and each token is given an ID. Using embedding matrix, these IDs are mapped to vectors named embeddings. During training phase, model learns the embedding matrix to represent the meaning of tokens from the input sentence. Transformer uses another vector to map positional encoding to input embeddings according to their positions in the sequence. Then several input sequences of same length are brought together. This process is called batching which allows Transformer to process multiple sequences in parallel.

The input sequences, along with their corresponding positional encodings, are then fed into the encoder. Every encoder layer has its own processing of input sequence using multi-head self-attention and feedforward processing. To mark the last input, model is given a token called start-of-sequence (SOS) token which the generation of the output sequence. This is the decoder input. Decoder also consists of same layers as the encoder, performing multi-head self-attention, encoder-decoder attention, and feedforward processing. At every step of the decoder, the model creates a probability distribution over the possible tokens in the output sequence. The token with the highest probability is chosen as the next output token. Model continues this process until the end-of-sequence (EOS) token is generated, indicating the end of generation. The output sequence enters the post-processing phase. The token IDs are converted back to text tokens.

Training phase of transformers is similar to generic DL models. The model is optimized to minimize a loss function which calculates the difference between the predicted output and the target. The optimization is performed using backpropagation and usually with stochastic gradient descent.

In general, the Transformer model has shown great advancement in wide range of NLP tasks. It was originally developed for NLP, but it has been successfully applied to time series data [78]. In the context of time series, the input of the transformer is no longer a

sentence but a sequence of observations over time. The fundamental difference of applying Transformers in time series lies in the way the input data is processed. In NLP, the input sentence consists of individual words, but in time series the input is not discreet, it is continuous-valued. To address this difference additional features are introduced. They include time stamp for each data point. These features are concatenated with the continuous values and then given to the transformer. An autoregressive prediction is also added to time series transformers. In autoregressive prediction, the output after every time step depends on the former outputs and inputs. The decoder takes the output sequence of previous time steps as inputs, and generates the following output.

Overall, using Transformers for time series related problems has shown promising results in a variety of applications, such as health care [37], climate science [63], finance [43, 53], and energy forecasting [85] and it is a promising area of research.

2.3 Evaluation Metrics

2.3.1 Cross Validation

Cross Validation refers to the technique for assessing the stability of the ML model. It is practiced for mitigating when the ML model fails to generalize, i.e., it suffers from overfitting. One of the most common practices is k-fold cross-validation in which the training dataset is split into k subsets. The ML model is trained on the k-1 subsets, and then validated on the left-out subset, see Figure 2.12. This process is repeated until the model evaluates all the subsets.

2.3.2 Confusion Matrix

Confusion Matrix is a matrix used for evaluating the performance of a classifier. The dimensionality of the matrix indicates the number of classes exists in the problem. For simplicity of explanation, we will analyze a binary classification problem which would give a 2x2 matrix, shown in Figure 2.13.

A confusion matrix includes 4 measurements that reflects the predicted values of the classifier and the actual values, i.e. labels, that comes from the input.

- *True Positive* (TP): prediction and label are positive.

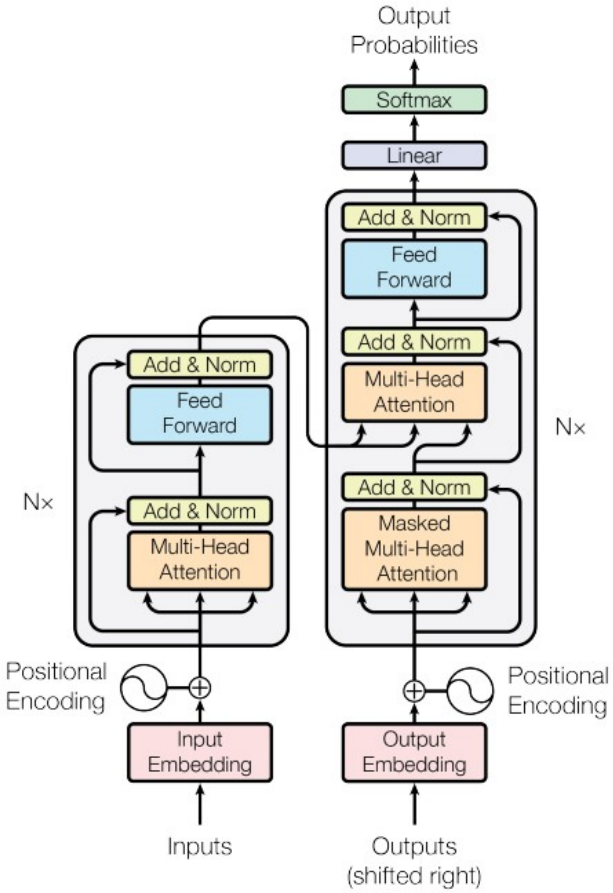


Figure 2.11: Transformer - model architecture. [74]

- *False Positive* (FP): prediction is positive, but label is negative.
- *False Negative* (FN): prediction is negative, but label is positive.
- *True Negative* (TN): prediction and label are negative.

Then, from these terminologies, we derive the following performance measurement methods:

- *Recall*: It shows the correctly predicted values from all the positive classes. It should be as high as possible.

$$Recall = \frac{TP}{TP + FN}$$

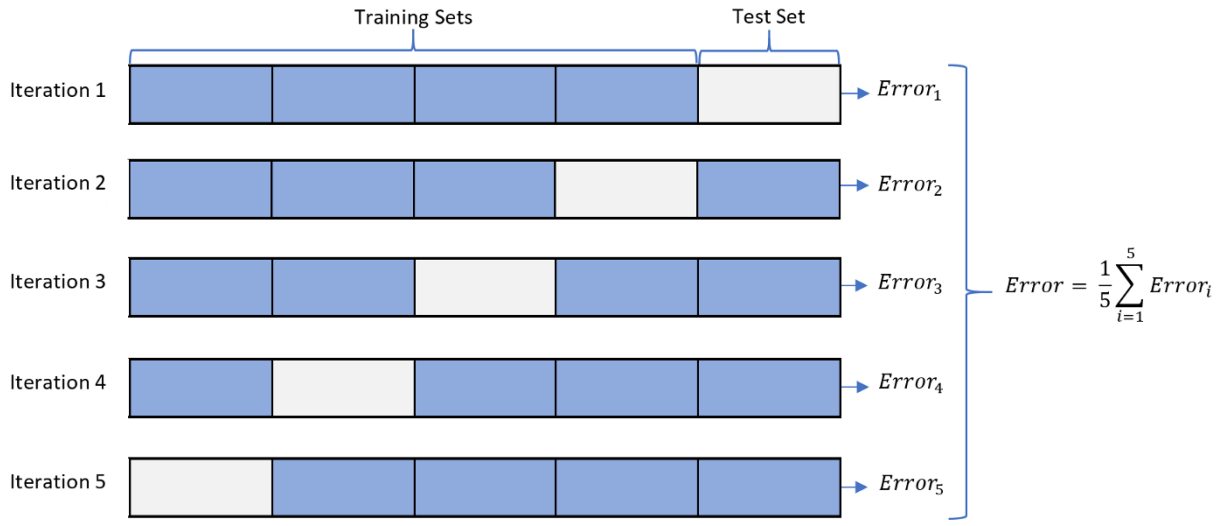


Figure 2.12: K-fold Cross-Validation

- *Precision*: It shows the actual positive values from all the positively predicted classes. It should be as high as possible.

$$Precision = \frac{TP}{TP + FP}$$

- *Accuracy*: It shows the correctly predicted values from all the classes. It should be as high as possible.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- F1 score: The challenge happens when the model has one of the precision or recall values being low and the other one being high. In this case, f1 score is used for measuring them. It should be as high as possible.

$$Accuracy = \frac{2 * Recall * Precision}{Recall + Precision}$$

2.3.3 Receiver Operating Characteristic (ROC)

Receiver operating characteristic (ROC) curve is a plot illustrating the classification ability of models at various thresholds. The ROC curve plots the true positive rate (TPR) on

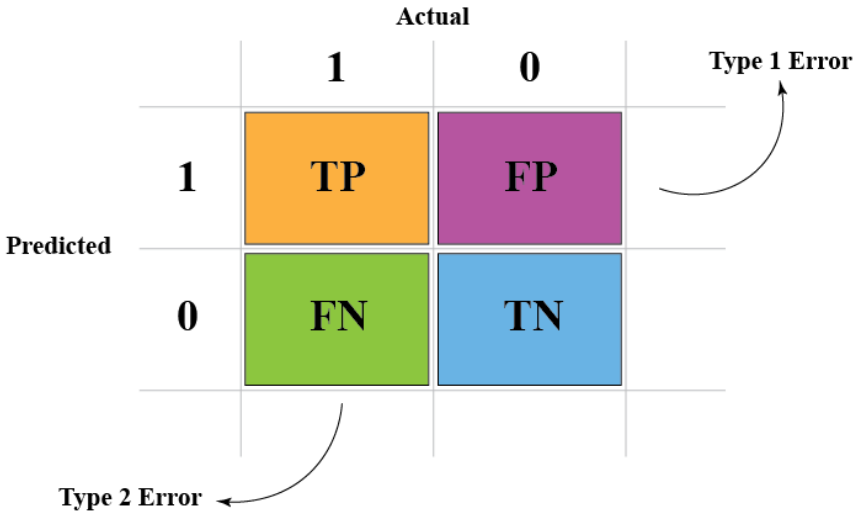


Figure 2.13: 2x2 Confusion Matrix

y axis against the false positive rate (FPR) on x axis, see Figure 2.14. TPR is another terminology for *recall*, see Equation 2.3.2 or *sensitivity*. FPR is calculated as follows:

$$Specificity = \frac{TN}{TN + FP}$$

$$FPR = 1 - Specificity$$

In order to compute the points in a ROC curve, we utilize a phenomenon called AUC. AUC stands for area under curve. It is the degree of separability which the model distinguishes a positive example from a negative example. The higher the AUC, the better the model is at separating between the two classes. A perfect classification model should have an AUC closer to 1. A mediocre model should have ROC at 45 degree making the AUC half the curve. It shows that half of the decisions the model makes are incorrect, i.e., the model cannot separate the classes at all. A poor model's AUC should be closer to 0 indicating that almost all the instances are misclassified.

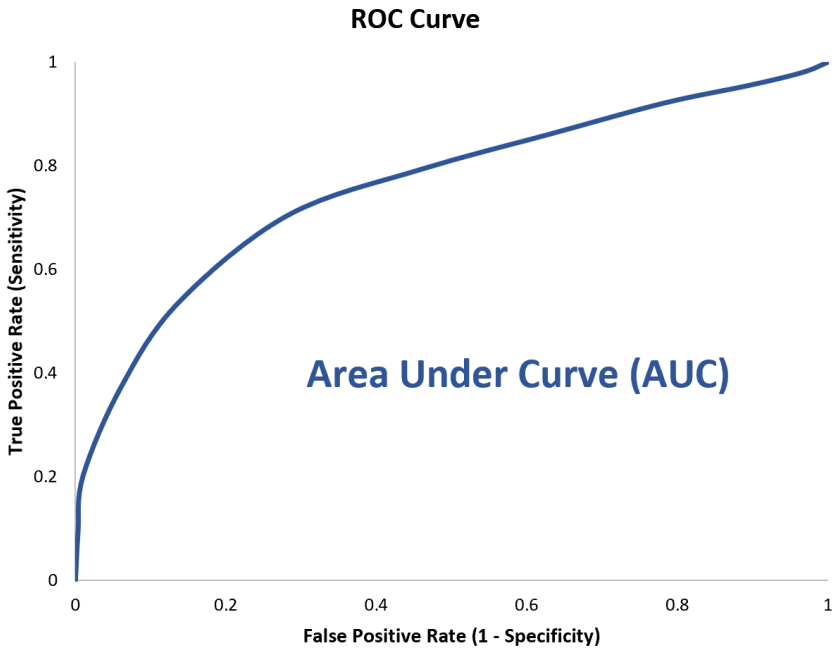


Figure 2.14: ROC curve

Chapter 3

Related Work

3.1 Network State Classification

Traffic classification is an essential aspect of network management, which involves identifying and classifying different types of traffic based on their characteristics and properties. This enables network administrators to allocate network resources effectively, prioritize traffic flows, and ensure that critical applications and services receive the required bandwidth and QoS.

In recent years, there has been a significant interest in developing effective traffic classification techniques that can accurately classify network traffic flows based on various parameters such as packet size, packet timing, protocol headers, payload content, and application behavior. This has led to the emergence of several approaches and techniques, including machine learning algorithms, statistical methods, and analytical techniques.

ML algorithms have shown significant promise in traffic classification to identify suspicious traffic [31, 54, 56, 73, 88], understand traffic flow [58], provide application-sensitive traffic classification [9]. ML algorithms algorithms use statistical models and pattern recognition techniques to learn from historical data and classify traffic flows based on various features and attributes. They have been applied in network traffic classification through supervised [17, 19] and semisupervised [29] learning methods.

In works [25, 34, 65] ML-based network status classification for IP networks have been studied. In [34], DT, SVM, and KNN are used to classify a software-defined wireless mesh-network status either congested or non-congested with 90–98% accuracies. ML models have been tested on two different datasets with observation periods of 5-60 seconds. In

Table 3.1: Performance and summary comparison for state-of-the-art state classification models.

Paper	**[[34]]**	**[[65]]**	**[[25]]**	**Our work**
Platform	SDN	Standard	NFV	Standard
ML-Type	Unsupervised	Supervised	Supervised	Supervised
Model	K-Nearest Neighbour	Naïve Bayes	Support Vector Machine	Extreme Gradient Boosting
Pro/Reactive	Reactive	Reactive	Both	Reactive
Contribution	Classify network status into congested or non-congested	Categorize the network status as faulty or normal and localize faults	Classify network status & predict the problem happened or not	Classify network status into 3 classes and localize faults

their work, KNN yielded the best performance on the datasets, 97.5% and 98.5% accuracies. In [65], a naïve Bayesian classifier is investigated to understand whether the network status is normal or faulty. In [25], ML and DL are combined into a hybrid framework for network status classification to assist network function visualization systems. The framework operates in two steps: (i) status classification and (ii) fault localization. If the network status is identified as faulty, then the framework localizes the fault, determines its root cause and severity. In their work, SVM detected the network states as faulty or normal (95.4% accuracy), and network faults as manifested (89.7% accuracy) or impending (95.1% accuracy); autoencoders predicted the severity of the impending faults (92%). In addition, to classify network status, while [57] preferred an analytical approach utilizing RTT values symbolizing network stability in 5 categories, [71] the SDN controller with statistical information fed as input, categorized the status as correct or faulty.

Previous works in the field of network status classification have generally divided the network status into two classes: *normal* or *faulty* where the faulty class typically indicates congestion in the network. Even works that have more than one class, usually subdivides the faulty network status again more congestion classes. However, in our work, we introduce a new class into the "faulty" status by further segregating the faulty class, diagnosing the underlying issue of the network status being faulty, as network physical issue or congestion where the former is caused by a physical issue that occurred somewhere in the network such as a device (e.g., routers) or link failure and the latter caused by heavy traffic that leads to deterioration in QoE. Moreover, we divide the faulty class into sub-states to localize the faulty device or the congested link.

Additionally, we have also compared XGB model, the state classifier, with the state-of-the-art models. Table 3.1 depicts a comparison between the proposed state classifier with the state-of-the-art models.

3.2 Traffic Prediction

Traffic prediction is an important component of traffic engineering, which involves predicting network congestion based on historical and real-time traffic data. Flow prediction, in particular, focuses on predicting the likelihood of congestion occurring on specific paths or flows within the network. This information can then be used to redirect traffic flows more efficiently to minimize congestion and optimize QoS and QoE for the end-user.

Various approaches have been studied to improve flow prediction to offer clients better QoS. DL models have been implemented to predict network traffic flows with high accuracy [47, 72], using techniques such as CNN [84] and LSTM networks [28]. ML algorithms have also been utilized for traffic prediction [12], such as DTs [41] and SVMs [55]. Additionally, big data analytics techniques have been applied to analyze large amounts of traffic data and identify patterns and trends to improve traffic prediction. In [30] big data analytic methods have been utilized for flow prediction in an SDN environment.

In an SDN environment, flow prediction can be achieved by analyzing network flow data and making predictions based on historical patterns and trends. By leveraging SDN's centralized control and programmable architecture, traffic can be dynamically routed to avoid potential congestion points and improve QoS for end-users. An extensive research on ML and DL based traffic classification and prediction in SDNs has been summarized in our previous work [50].

3.3 Failure Recovery

Restoring a network failure is a crucial aspect of network management that requires a well-planned fault management process. This process usually involves four key steps: detecting the failure, diagnosing the root cause, alerting the responsible parties, and resolving the issue.

Traditionally, MPLS fast reroutes were used to address fault management problems. In [1] and [3], failed links are avoided by building a tunnel from the start point to the end point of the broken path, effectively bypassing the failed link.

However, with the rise of SDN, fault management has taken on a more proactive approach. In [32, 64, 75], SDN controllers predict the occurrence of congestion or link failure and propose restoration algorithms. Similarly, in [39], a hybrid proactive and reactive system within SDN is used to build backup paths that can address the needs of different applications.

Other works have explored alternative approaches for handling network failures. For instance, [59] calculates rerouting paths using the shortest-path spanning tree algorithm, while [42] uses asymmetrical routing to set the routes.

In addition to these approaches, [15] addresses multi-path link failure by directing traffic through less congested links. Meanwhile, [69] proposes flow table prioritization in the switch to offer backup paths that can mitigate single link failure. Overall, these works demonstrate the importance of fault management in network management and offer various approaches that can be taken to address the issue.

3.4 Traffic Engineering

Traffic engineering is a crucial process in computer networks, where engineers manage the flow of traffic through the network to achieve predefined objectives, such as minimizing packet loss, optimizing bandwidth utilization, and reducing network latency. Various methods have been proposed in industry and academia to achieve these goals, including multipath routing [27], source and segment routing [20], distributed routing protocols [23], smart routing techniques [45], packet scheduling with queue setting [80], energy-aware routing and flow scheduling [10, 36], and elevator scheduling based load-balancing [38].

Recent advancements in SDNs have further revolutionized traffic engineering, allowing for more effective network management. SDN's controller regulates traffic flow and designs reasonable routing paths to improve network resources [71]. The distinguished characteristics of SDN such as separating control and data plane enables programmability in administering network behavior. SDN's controller implements a sequential DL model in [87] and an unsupervised ML model in [35] to execute path-planning.

However, the current works on traffic engineering have limitations. They often focus on only one aspect of traffic management, such as traffic prediction, failure recovery, or traffic engineering. Moreover, they do not consider the OPEX costs associated with network traffic management. Finally, they often classify network status into two categories, normal or faulty, which can be insufficient to diagnose and address the underlying causes of network problems.

Our work is a unification of all these fields. We address these limitations by classifying the network status into three categories and proposing solutions that minimize OPEX costs while effectively managing network traffic. Our approach includes a fault management system that comprises four steps: detecting, diagnosing, alerting, and resolving. By taking

this comprehensive approach, we aim to optimize network performance while minimizing the associated costs.

Hence, our work is closely related to [71] and [35]. In [71], in faulty network status, future traffic behavior of the network is predicted and a traffic scheduling algorithm is created to avoid congestion to reoccur. In [35], the ML model knowing the current network state, predicts the best path.

3.5 Transformers in Time Series

The Transformer architecture has proven to be effective in sequence processing tasks such as NLP without using RNNs. It uses attention mechanisms and has shown promising results in other areas such as images, point clouds, video, audio, and time series forecasting. The effectiveness of utilizing Transformer-like models on time series data is being investigated in study [18]. In this work, it is concluded that unlike RNN based models such as LSTM and GRU, transformers offer effective solutions for long-term dependencies in time series data.

Yuan and Lin employed transformers as a means of mitigating overfitting in the scenario of inadequate labeled data to train NNs [82]. The study proposes a self-supervised pretraining scheme for satellite image time series classification using a transformer-based network. The scheme learns general-purpose spectral-temporal representations from large-scale unlabeled data and improves classification accuracy with fine-tuning using small-scale labeled data. Experimental results demonstrate its effectiveness.

The paper [40] introduces the Temporal Fusion Transformer (TFT), a deep learning model for multi-horizon forecasting that combines high performance with interpretable insights into temporal dynamics. TFT uses specialized components to handle static covariates and observed inputs effectively, including sequence-to-sequence and attention-based temporal processing components, static covariate encoders, gating components, variable selection, and quantile predictions. The authors demonstrate significant performance improvements over existing benchmarks on a variety of real-world datasets and highlight three practical interpretability use cases of TFT.

In [81], authors proposed Multivariate Time-Series Imputation with Transformers (MTSIT), an unsupervised transformer-based method for imputing missing values in multivariate time series. MTSIT uses the encoder part of the transformer and trains an autoencoder to jointly reconstruct and impute stochastically-masked inputs. Experimental results demonstrate that MTSIT outperforms state-of-the-art imputation methods. Future work

can explore incorporating confidence bounds in a Bayesian setting, designing advanced MTSIT models, and using confidently imputed points to improve performance on forecasting and classification. Additionally, MTSIT can be extended to recent efficient transformer variants.

In our work, we applied state-of-the-art transformer model on our dataset. To the best of our knowledge, our work is the first to utilize transformer models for OPEX aware network traffic optimization without network state classification. The results obtained from the experiments are promising.

Chapter 4

System Design

Collecting data is the very first step -and an important part- of building any ML model. In the case of network traffic management, having a comprehensive and diverse dataset is critical for training and evaluating the effectiveness of ML and AI models. Unfortunately, to the best of our knowledge, there is no open-source network dataset that is specifically designed for our experiments. Therefore, we had to create our own testbed and generate our own dataset using the GNS3 emulator.

To implement our testbed, we used a combination of physical and virtual devices to simulate a real-world network environment. Our physical devices included routers, switches, and servers that were connected using Ethernet cables. We also used virtual devices, such as virtual routers and virtual machines, to create a more diverse and complex network environment.

We used the GNS3 emulator to simulate various network scenarios and generate network traffic. GNS3 allowed us to emulate different types of network devices and run real software on virtual devices. We generated traffic by running various network applications, such as web browsing, video streaming, and file transfer, on the virtual machines in our testbed. We captured network traffic using Wireshark, a popular network protocol analyzer, to create our dataset.

The collected data was preprocessed to prepare it for training and evaluation of the ML and AI models. We cleaned the data, removed any duplicates or irrelevant information, and normalized the data to ensure that it was in a consistent format. We then split the data into training and testing sets to train and evaluate our models.

Our system design for managing network traffic includes two approaches: stateful and stateless ARE. Both approaches are designed using a rule-based system that prioritizes

network traffic based on predefined rules. The only difference is that stateful ARE uses network state as one of the metrics in the dataset to recommend actions whereas stateless ARE recommends actions from raw data. In the upcoming sections, we will explain the implementation details of these two approaches and the results of our experiments.

4.1 Testbed Implementation

GNS3 [2] is a widely used open-source software heavily employed in the industry by companies such as Google, NASA and, Pfizer. We chose to use GNS3 as an emulator tool to setup set up our network topology, as shown in Figure 4.1. The topology was designed to represent a realistic ISP networkwith multiple paths to clients and content servers [83]. We carefully balanced the complexity of the topology, ensuring that it was not too simple to be unrealistic but not too complex that it was impossible to implement expert rules since they are of utmost importance to compare ARE's performance to the expert rules. Hence, in our design, we balanced these two constraints. The topology consisted of three Autonomous Systems (ASs): AS-1 is the internal network whereas AS-2 is the external. The internal network is where ISP network provides services to the clients. AS-3 is where FTPs, i.e. the content servers, are placed. AS-2 is where a neighboring ISP connects to the internal AS. Since AS-2 is an external ISP, routing traffic through this network is costly compares to forwarding packets internally. AS-2 symbolizes a backup network to be utilized when R2 breaks down or Link 2-4 gets congested.

It is worth mentioning that DASH video streaming was first tested in the testbed. However, and since we are using Cisco community edition router images. The router bandwidths have drawbacks, this issue is further discussed in the next paragraphs. DASH videos require a considerably large bandwidth to reflect different QoE levels. We concluded that there are the two solutions. First is to render the videos in very low quality and configure DASH server and its client to measure QoE based on the newly rendered video or search for an alternative that can serve the purpose of data transmission but doesn't require the liability of adjusting and re-configuring DASH. as a consequence, we opted to use FTPs due to its ease of usage and transmission limit controllability.

In order to simulate the real-life situation, we introduced a billing system indicating the cost of selected routes. Hence, there are three paths from every client to the content servers. PATH_1 and PATH_2 are within AS1, making them internal routes which are less costly whereas PATH_3 is the external route. PATH_3 directs the traffic to the content servers in AS3 through AS2. Since AS2 extends its services to AS1, it charges more. Also, fixing any issues happening on this route consumes resources such as time and

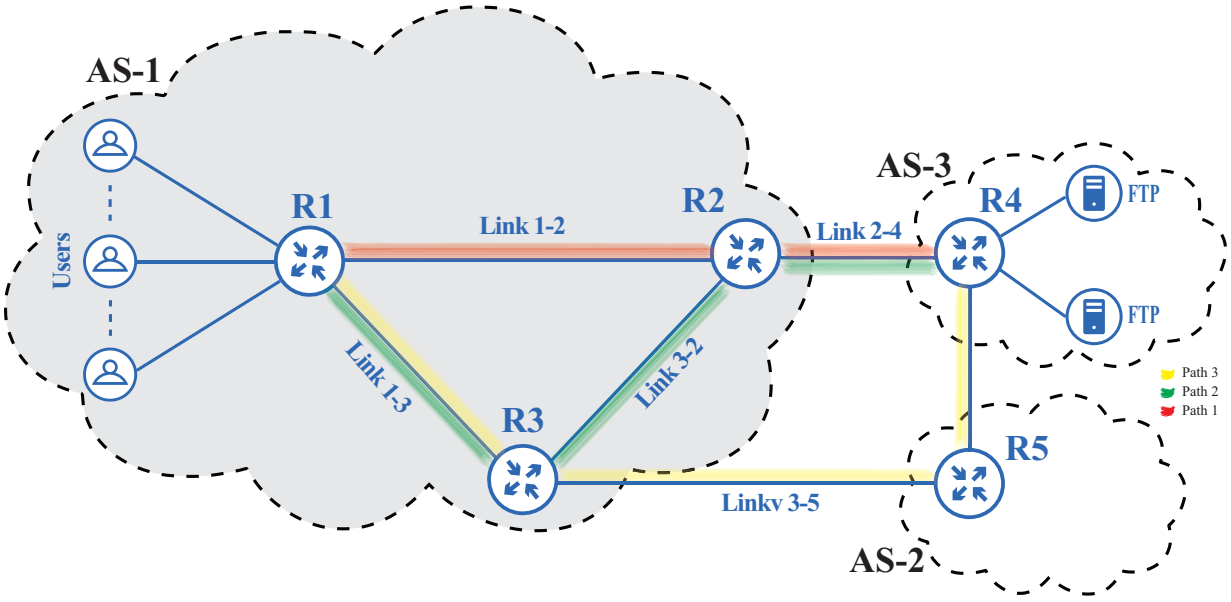


Figure 4.1: Network Topology.

money which makes PATH_3 the most expensive route. PATH_1 is the cheapest option because it is the internal route with the minimum number of hops. PATH_2 is cheaper than PATH_3 because it is an internal path but more expensive than PATH_1 because it has more number of hops. The cost of each paths are calculated taking into considerations of variables presented in Table 4.1 and shared in Table 4.2. To make our design more realistic, we also added a reward and punishment system with regards to the end-users' QoE per path, see Table 4.1. This table numerically represents the core definition of network scenarios implemented in the experiments and the values are provided by our industrial partner Ciena's NOC team. Overall, our aim is to perform network status classification on AS-1 and recommend the best available action to help restore the network stability in OPEX aware manner.

In our testbed, we used Cisco's IOSv images to model network devices like routers and switches. These images were chosen because they are widely used in industry and have similar features to physical devices. In addition, we used containers to model headless Ubuntu 18.04 OS images. These images are modeled by containers because they are lightweight and provide standalone environment that is isolated from the rest of the system [46].

To ensure that the content servers, the headless Ubuntu 18.04 OS images, were isolated from external network conditions, we placed them on the testbed machine. Clients, on the other hand, were placed on AS1. The network connectivity between the devices in

Method	Billing
Each hop	-1
External AS	-5
Fix Network	-5
TE (re-routing)	-2
Bad QoE	-5
Medium QoE	0
High QoE	5

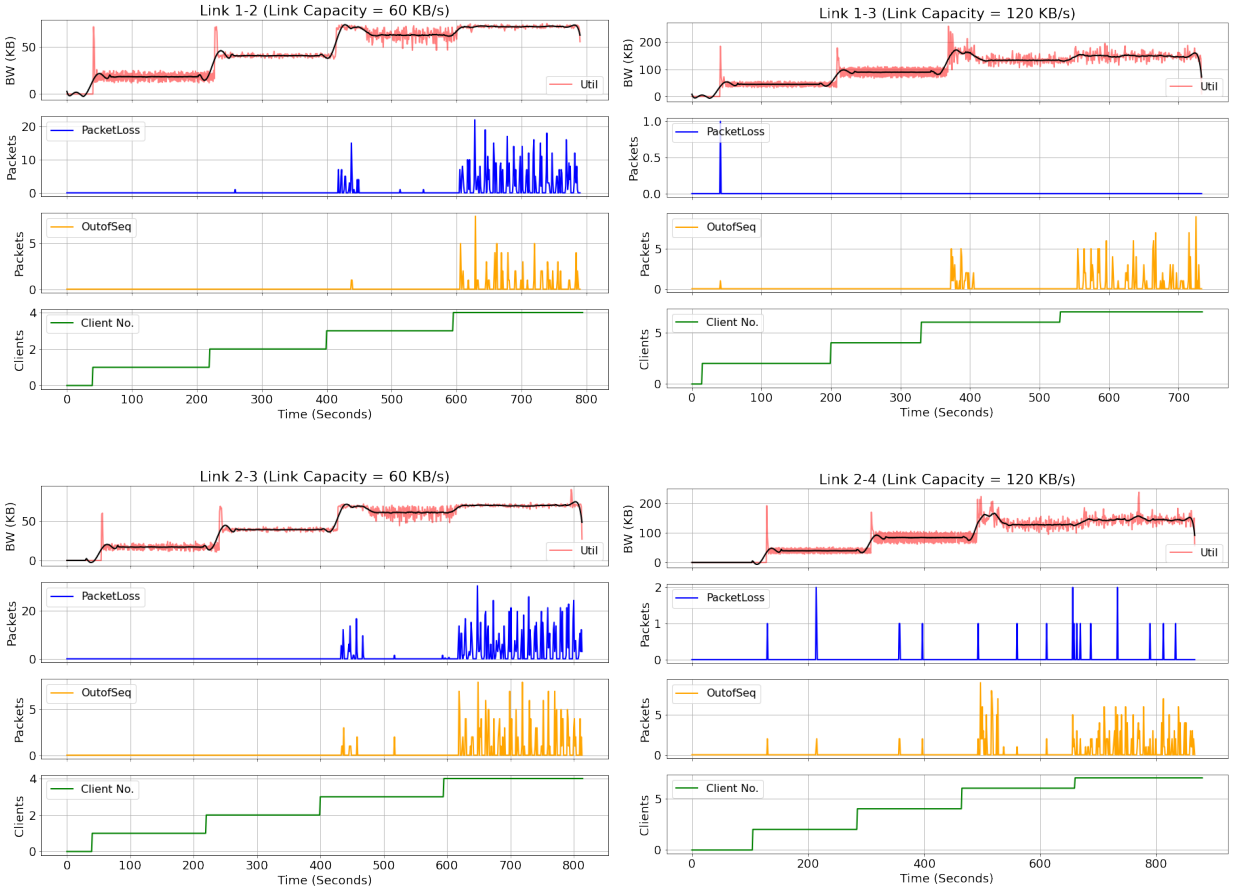
Table 4.1: Billing System

Path	Cost
PATH_1	OPEX = 2 IGP hops × 1 = -2
PATH_2	OPEX = 3 IGP hops × 1 = -3
PATH_3	OPEX = 2 IGP hops × -1 + -5 (AS2) = -7

Table 4.2: Path OPEX costs

our testbed was established using the Open Shortest Path First (OSPF) routing protocol. This protocol was chosen because it is widely used in enterprise networks and is known for its fast convergence time and scalability. In order to create paths between the clients and content servers, we used Multiprotocol Label Switching (MPLS) over OSPF. MPLS is a protocol that uses labels to direct traffic along predetermined paths. This allows for faster and more efficient routing of packets, as the labels can be used to bypass complex routing decisions. In order to control which clients connect to which path, we used Access Control Lists (ACLs). An ACL is a set of rules that can be used to filter traffic based on criteria such as source IP address, destination IP address, or protocol type. By using ACLs, we were able to ensure that clients were directed to the appropriate path based on their requirements and the available network resources. In summary, ACLs controlled the paths connecting users and content servers, deciding which client lands on which path. In the end, our testbed was designed to closely mimic a realistic ISP network, while also being flexible enough to allow for the testing of different scenarios and configurations where we can experiment with ARE.

The modeling of network devices in our testbed was done by utilizing Cisco's IOSv images as they are currently the only available open-source option. However these images carry a significant limitation as they are not designed to support high bandwidth utilization. The maximum bandwidth they can support is around 2 Mb/s, as they were primarily



(a) Unique Links. (b) Shared Links.

Figure 4.2: MPLS Link Utilization.

created for network algorithms and configurations. Since our end-users were transferring files using the TCP windowing algorithm [8] and occupying all available bandwidth, we had to apply layer-3 policies to overcome this limitation. We restricted the file transfer speed to only 20 KB/sec and set a threshold of 20 KB/s for each user using the policies shown in Table 4.3. This ensured that no more than 3 clients could transfer files simultaneously at any given time. Wireshark was used to verify that these policies were correctly implemented. This also provided that when more than 3 clients are connected to the link, that link will be congested, see Figure 4.2. We also categorized the links in our testbed into two types: Unique links and Shared links. Unique links refer to links with only one path. Hence its maximum bandwidth is

$$20KB/secx3(clients) = 60KB/sec$$

While Shared links refer to links that combine more than one path. Therefore its maximum bandwidth is twice as much of the unique link. By utilizing the layer-3 policies to identify the number of clients connected to each path, we could determine the network states in our state classifier. In this way, we defined the network states as;

- *normal*: when every link has maximum three end users.
- *congestion*: when more than three end users are connected to a link.
- *network physical issue (NPI)*: when any network device is broken or there is path disconnectivity. In this case, the delay and jitter should range between 80–350 and 5–60 ms, respectively.

Table 4.3: Layer-3 Policies Summary.

Link	**Type**	**Max BW**	**Member**	**Policy**
Link 1-2	Unique	60	Path1	R2: G0/0 peak 60
Link 2-4	Shared	120	Path1&2	R4: G0/1 peak 120
Link 1-3	Shared	120	Path2&3	R3: G0/0 peak 120
Link 2-3	Unique	60	Path2	R2: G0/1 peak 60
Link 3-5	Unique	60	Path3	R5:G0/0 peak 60

This allowed us to accurately analyze the network performance and make recommendations to restore network stability in an OPEX-aware manner.

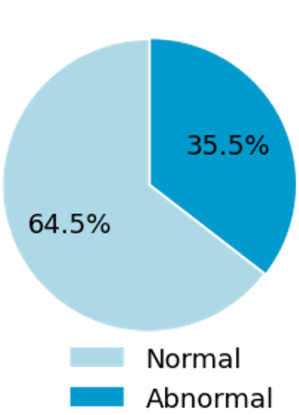
4.2 Data Collection and Preparation

This section provides a detailed explanation of the mechanism used for collecting state and action data. We developed an API in Python to communicate with the GNS3 emulator, which allowed remote control of network elements such as routers and docker containers. This API enabled us to remotely turn these elements on and off at any time, perform live configurations on them, and even impose delay and jitter on any link in the network. In addition, the API allowed us to control any operation on the end users.

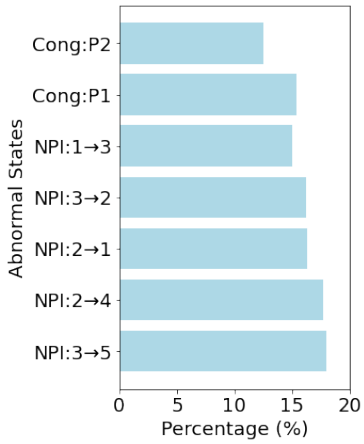
The data collection process began by randomly connecting each end user to a path and then randomly introduced link disconnection or link congestion problem. The script collects QoS and QoE metrics every 30 seconds during this process. The QoE metric was measured by downloading speed only, while the QoS metric included end-to-end delay, jitter, and packet loss values, i.e., SLA measurements of the three paths, as well as input and output rates and packet loss values of the router ports in AS-1. The QoS and QoE metrics were then united by aligning their timestamps. Table 4.4 presented the features and their associated labels in the dataset.

Data collection took over 20 days and resulted in 56,000 data points. Two algorithms, Algorithm 4.1 and Algorithm 4.2, were run side by side during this process, i.e. the output of Algorithm 4.1 was fed into Algorithm 4.2. Algorithm 4.1 was used to categorize the network state, while Algorithm 4.2 was used live to decide which action to take by mimicking NOC rules. We noted that it was simple and straightforward to define the rules presented in Algorithm 4.1 due to the simplicity of our testbed. However, as the network becomes more complex, it becomes increasingly difficult to define these rules, and NOCs hire expert technicians. According to Algorithm 4.2, ARE is conditioned to route client traffic internally before directing them to an external network, prioritizing the internal network over the external network since routing internally is cheaper with regards to OPEX preferences.

The dataset ultimately consisted of 66 different data features. The labeled state and action distributions are shown in Figure 4.3 and Figure 4.4, respectively. We observed that the Normal state dominated the state classes, see Figure 4.3(a), reflecting real-life situations, as Abnormal states (Congestion and Network Physical Issue) are discrete events. This was achieved due to the policy we followed, by resetting the network to the Normal state after every time one of the Abnormal states was introduced to the network. For example; if at time t PATH_1 is congested by adding an extra client to the link, then at time t+3, one of the clients on PATH_1 is relocated to either PATH_2 or PATH_3 so that the network state could return to its normal state. Likewise, if at time t, the current state is physical issue, then at time t+3, the filter is removed to enable the network to become Normal again. Similar phenomenon can be observed in Figure 4.4. The Do Nothing action is dominating the actions space since the corresponding action to the Normal state is Do Nothing. As a consequence of this strategy, we faced the imbalanced classification problem; i.e., one class having larger number of data points compared to the other classes in the dataset. This is an intended outcome of the adopted strategy to realistically reflect the real world. Class imbalance problem can be tackled via various methods such as upsampling, downsampling, using ensembling methods, etc. In this work, we preferred to preserve the data distribution. Hence, we approached the problem by utilizing ensemble methods.



(a) Class Distribution.



(b) Minority Class Breakdown.

Figure 4.3: States space distribution.

Figure 4.3(b) lists the percentage distributions of the labeled abnormal states which are denoted as NPI (Network Physical Issue) and Cong. (Congestion). Figure 4.4(b) lists the percentage distributions of the recommended actions by the expert rules which are denoted as FIX (fixing the faulty device by restarting or re-configuring through API) and TE (rerouting the end users). The NOC opts the best available action which gives the minimum cost to reroute the end user.

During the training phase of the ML models, we realised that the 30-second wait time period was not enough to clearly distinguish the congestion states. This is why we formed a selection from two separate parts. We aggregated the congestion states by taking the average, maximum, or minimum of SLA and QoE measurements. In the case where we have one Normal state and one Abnormal state, we opted the Abnormal state. Hence, we set the Abnormal state as the deterministic variable. The aggregation of the dataset is done as follows. For example; the aggregation of RTT measurement is done separately for RTT_{min}, RTT_{max}, and RTT_{avg}. For RTT_{min}, we take the global minimum:

$$RTT_{min_{agg}}[T] = min(RTT_{min}[T-1], RTT_{min}[T])$$

Similarly for RTT_{max}:

$$RTT_{max_{agg}}[T] = max(RTT_{max}[T-1], RTT_{max}[T])$$

and for RTT_{avg}:

$$RTT_{avg_{agg}}[T] = avg(RTT_{avg}[T-1], RTT_{avg}[T])$$

Table 4.4: Features and Labels Summary.

Features

QoS (57 features):

- Per-Path (30 features; 3 Paths × 10 features):
  - Delay: End-to-end Path delay (min/avg/max).
  - Jitter: End-to-end Path jitter (min/avg/max).
  - Packet loss: End-to-end Path packet loss (min/avg/max).
  - Out of Sequence: In-ordered packets.
- Per-Port (27 features; 3 routers × 3 Ports × 3 Features):
  - Drops: Packet Drop count for ports 0, 1, and 2.
  - Input Rate: Ingress packet rate for ports 0, 1, and 2.
  - Output Rate: Egress packet rate for ports 0, 1, and 2.

QoE features (9 features; 3 Paths × 3 features):

- Download speed: FTP transfer speed (min/avg/max).

Labels

Reroute due to congestion:

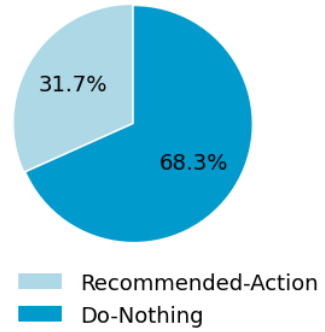
- TE-P1→P2: Reroute clients from path 1 to path 2.
- TE-P1→P3: Reroute clients from path 1 to path 3.
- TE-P2→P3: Reroute clients from path 2 to path 3.

Reroute to optimize cost:

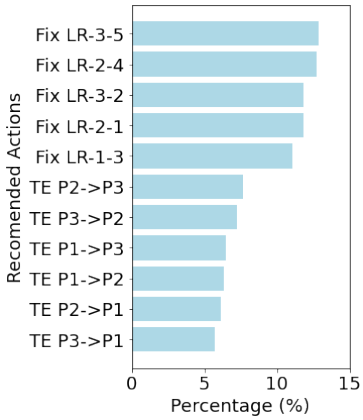
- TE-P2→P1: Reroute clients from path 2 to path 1.
- TE-P3→P1: Reroute clients from path 3 to path 1.
- TE-P3→P2: Reroute clients from path 3 to path 2.

Fix device or link issues:

- Fix LR-1-3: Fix issues between router 1 and router 3.
- Fix LR-2-4: Fix issues between router 2 and router 4.
- Fix LR-2-1: Fix issues between router 2 and router 1.
- Fix LR-3-2: Fix issues between router 3 and router 2.
- Fix LR-3-5: Fix issues between router 3 and router 5.



(a) Class Distribution.



(b) Minority Class Breakdown.

Figure 4.4: Actions space distribution.

Algorithm 4.1 Operational labeling rules for this experiment.

```
while True do
  choice = random(add_client, physical_prob)
  if choice == physical_prob then
    label = NPI
  else
    if no. of clients in (PATH_1)>3 then
      label = Congest:P1
    else if no. of clients in (PATH_2)>3 then
      label = Congest:P2
    else
      label = Normal
    end if
  end if
end while
```

Algorithm 4.2 Operational rules mimicking NOC actions for this experiment.

```
Input: Network State
Output: Action
while True do
  if Network State == Congestion then
    if Internal Path Available then
      Action=Reroute Internally
    else
      Action=Reroute Externally
    end if
  else if Network State == Network Failure then
    Action=Fix the physical device
  else
    Action=Do Nothing
  end if
end while
```

The action for these aggregated data also set as the similar approach in state labeling. The abnormal action, i.e., any action other than *Do Nothing*, such as reroute or fix, is set to be the deterministic action. This finalized the dataset and experiments started.

During our experiments, we noticed a significant challenge related to scalability. If we increase the size of our network architecture, the number of data features and classes also increase. This, in turn, creates a bottleneck during the training of NNs, especially in the context of network status classification tasks. The problem of scalability is a common one in ML and is often related to the computational requirements of training larger models. In the case of our network status classification task, the sheer size of the network architecture we were working with meant that we were dealing with a large amount of data, which made the training process time-consuming and resource-intensive.

To address this challenge, we needed to find a way to overcome the limitations of network state classification. We recognized that it was not always necessary to accurately classify the current state of the network to take meaningful actions. Instead, we focused on developing a set of recommendations that could be made without relying on network state classification. We also found that by using transformer models, we were able to better handle the complexity of the data and the network architecture. Transformer models are a type of NN architecture that is particularly well-suited to handling large amounts of data, making them an ideal choice for our needs.

By leveraging the power of transformer models and focusing on actionable recommendations, we were able to overcome the challenge of scalability and achieve our goals for the project. This experience highlighted the importance of carefully considering the tradeoffs between accuracy and scalability when working with large datasets and complex models.

4.3 Stateful Action Recommendation Engine

The development of an AI model capable of classifying network state and localizing faults is essential in maintaining a stable and efficient network. This is why our experiments aimed to collect various metrics, including QoS, QoE, and per-device metrics from the testbed, to develop a model that can provide recommendations for the best action to take to address any network issue. To collect these metrics, we used a testbed that simulated a real-world network environment. In this testbed, we measured QoE through the download speed in FTPs, while QoS was measured through per-path and per-device metrics. These metrics included SLA measurements such as end-to-end delay, jitter, packet loss, out-of-sequence cases, and discarded samples from Path1, Path2, and Path3 for per-path data. Meanwhile, for per-device metrics, we measured port input and output packet rates and AS-1 router packet loss. Collecting these metrics is crucial in developing an AI model that can accurately classify the network state and recommend the best course of action. The QoE metrics help determine the user's experience, while QoS metrics provide a more in-depth view of the network's performance. By combining these metrics, we can develop a comprehensive model that can classify network state and localize faults more accurately for us to offer the best action available. Additionally, collecting these metrics enables us to understand the network's behavior and identify any recurring issues that may arise. This information can help us improve the network's overall performance, as we can identify and address the root cause of any network issues.

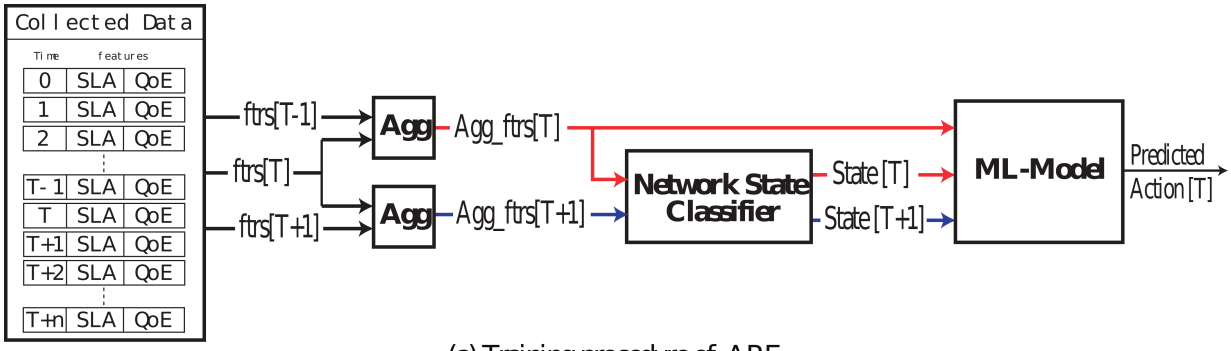
QoS, QoE, and SLA measurements are collected from the testbed. Once the data was collected, it was subjected to a preprocessing phase. The preprocessing phase is a crucial step in any ML project because it involves cleaning the data, formatting it, and making it ready for use by the ML model. Since every project is unique, the data needs to be tailored to the problem and to the proposed solution. In our experiment, the collected data was concatenated, normalized, and aggregated. Concatenation refers to combining all the relevant data sets into one. Normalization is the process of adjusting the values of different variables to a common scale so that they can be compared and analyzed together. Aggregation, on the other hand, involves combining the data into a summary form that is easier to analyze. Once the data was preprocessed, it was split into three sets: training,

validation, and testing sets. The training set was used to train the ML model, while the validation set was used to evaluate the performance of the model during training and adjust its hyperparameters. The testing set was then used to assess the final performance of the model. The ML model was then trained and validated using the training and validation sets. There are different types of ML models, but in our experiment, we used DT, GB, XGB, NN, and transformers were used. The models were then used to predict the network state. The network state classifier provided a list of probabilities of having state Normal, Congestion, or Network Physical Issue. This created the network state classifier [48] and the dataset distribution has the following format: normal 15%, congestion 81%, physical problem 4%.

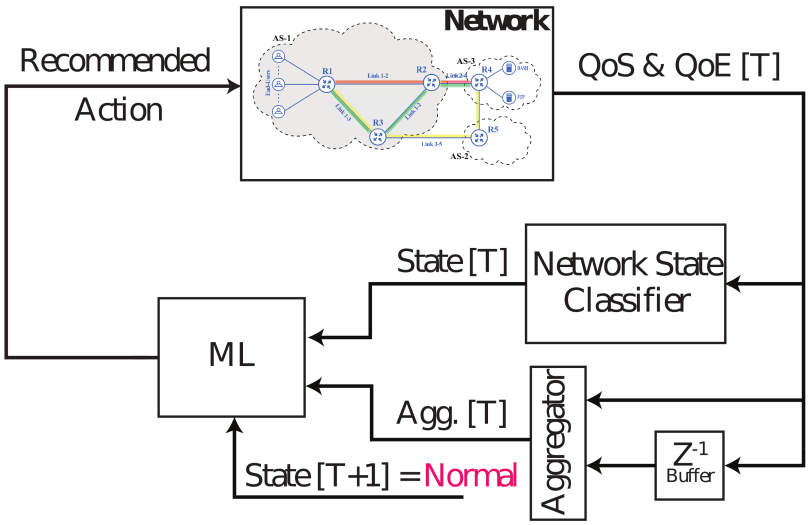
After the network state classifier was trained, the next step was to integrate it with the network architecture to create a closed-loop system. This integrated system would enable the automatic delivery of predicted actions on the network. To achieve this, the state probabilities of the network state classifier at time $[T]$ and $[T+1]$ and the aggregated features at time $[T]$ were given into the ML model as inputs. Additionally, the action label is the action at time $[T]$. This enabled the models to forecast which action altered the network state at time $[T]$ to the network state at time [T+1]. The training methodology for this process is depicted in Figure 4.5a. After the action recommender training process, the ML model was merged with the network architecture to create a live closed-loop system. This closed-loop system is an integrated system that automatically delivers predicted actions on the network. As seen in Figure 4.5b, at time $[T]$, QoS and QoE measurements are given to the Network State Classifier to create the state probabilities vector. At the same time, QoS and QoE measurements are aggregated with the measurements generated at time $[T-1]$, denoted by Z^{-1}. Then, the state probabilities vector and the aggregated measurements at time $[T]$ are given as inputs to the ML model. Meanwhile, the state probabilities vector at time $[T+1]$ is consistently set back to the Normal state to train ARE to learn to bring the network to Normal state. Finally, the python-based script implemented the predicted action automatically.

4.4 Stateless Action Recommendation Engine

The solution to the performance degradation of Stateful ARE was the creation of a new model called *Stateless ARE*. This new model did not require a network state detector, which could prevent the introduction of misclassification errors and avoid error propagation in the system. With the elimination of the state detector, the cascaded design of ARE was also eliminated, which can add delays to the system and be computationally expensive.



(a) Training procedure of ARE.



(b) ARE's Closed-loop diagram.

Figure 4.5: Action Recommender Training Methodology for Stateful ARE

Stateless ARE relies on the fact that the aggregated features at time $[T]$ contain all the necessary information to make a decision on the next best action. Therefore, instead of using the state probabilities vector, i.e. adding new data into the dataset, the model uses the aggregated features at time $[T]$ and the action label at time $[T]$ to predict the network's next state. This design allows the system to bypass the state detection process, reducing the computational complexity and training time.

The performance evaluation of Stateless ARE was compared to that of Stateful ARE using XGB model, the best Stateful ARE model, see Section 5.1 for details. The results

Action Category	Action
general	Do Nothing
reroute due to congestion	reroute clients from path 1 to 2
	reroute clients from path 1 to 3
	reroute clients from path 2 to 3
reroute to optimize OPEX	reroute clients from path 2 to 1
	reroute clients from path 3 to 1
	reroute clients from path 3 to 2
fix network physical issue	fix issue between router 1 and 3
	fix issue between router 2 and 4
	fix issue between router 2 and 1
	fix issue between router 3 and 2
	fix issue between router 3 and 5

Table 4.5: Stateful ARE's Recommended Actions

showed that Stateless ARE outperformed Stateful ARE, as it closely followed the NOC Mimic curve with no significant degradation over time. The performance of Stateful ARE started to deteriorate around iteration 770, while Stateless ARE maintained a consistent accuracy rate.

One advantage of Stateless ARE is that it is scalable, making it suitable for large networks. Adding a new link to the network only requires retraining the model with the new input, which is a much simpler and faster process than retraining the entire model in Stateful ARE. Additionally, it is easier to monitor and maintain from an MLOps viewpoint because it only consists of one ML model instead of two, i.e. no cascaded ML models. Furthermore, the elimination of the state detection process allows for faster automation. The system can make decisions and take actions in near real-time, which is crucial for critical network applications. Stateless ARE reduces the delay between the input data and the predicted action, which is particularly important in dynamic network environments.

In summary, the design of Stateless ARE is similar to that of stateful ARE, except for the removal of the network state detector. The AI model can be fed directly from the matrix with raw data, making the system more efficient and less prone to misclassification errors. Figure 4.6 depicts the design of Stateless ARE. We can conclude that Stateless ARE is a viable solution to address the performance degradation of Stateful ARE caused by misclassification errors. It eliminates the need for a network state detector, which simplifies the model and reduces the computational complexity and training time. Moreover, it is scalable, easier to monitor and maintain, and can operate in near real-time, making it

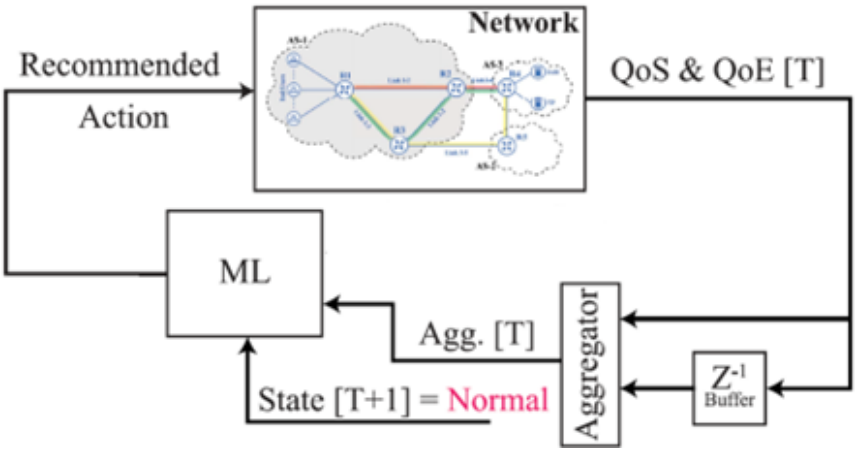


Figure 4.6: System design of Stateless ARE.

suitable for critical network applications.

4.5 Transformers

The removal of the network state detector eliminated misclassification errors introduced within and refrained the system from a phenomenon called error propagation. The cascaded design of Stateful ARE added delays in the system because the second ML model had to wait for the first ML model to produce its output before it could start its own training process. Depending on the network size, this process could be extremely time-consuming during training process. Therefore, eliminating the state detection process accelerated automation, and maintenance of two cascaded ML models was computationally expensive and required complex monitoring from an MLOps viewpoint. We overcame these challenges with the creation of Stateless ARE. The design of Stateless ARE is almost the same as Stateful ARE, but without the network state detector, allowing the AI model to be fed directly from the matrix with raw data. However, the DL model for Stateless ARE did not outperform the Stateful ARE. It yielded almost similar results. To advance the performance of Stateless ARE, we utilized transformers. The use of transformers in the Stateless ARE allowed us to improve the performance of the model and also apply it to real-life scenarios.

Transformers are a new type of DL model that were originally designed for NLP tasks, but have since been successfully applied to other problems as well. We modified the original transformer design, removed the decoder stack from the original transformer architecture

and replaced it with 4 linear fully connected NNs. Then we applied it to our time series data. The input to our transformer model consists of a set of features derived from the original dataset. The model output consists of 17 classes that correspond to different actions that can be taken to remedy network failures or improve traffic engineering. Training the transformer model mandated the deployment of a sole NVIDIA RTX 3090 graphics processing unit, resulting in a cumulative training period of four hours, a duration that may be considered extensive when compared to the training periods required for alternative models, such as DT, which typically occur in mere seconds or minutes.

To process the input data, we first apply positional encodings to the input features and then pass them through the encoder component of the transformer model. The encoder is responsible for processing the input data and producing a set of hidden representations that capture the key temporal dependencies and patterns in the data. We choose the default number of attention heads as 3, as suggested in the seminal work by Vaswani et al [74]. The attention mechanism enables the model to selectively attend to different parts of the input sequence and to assign different weights to different elements in the sequence based on their importance for the task at hand. The decoder component of the transformer model consists of a FCNN, i.e. a vanilla neural network, that maps the hidden representations generated by the encoder to the output classes. ReLU activation functions are used in the decoder layer, which have been shown to be effective in enhancing the non-linearity of the model and preventing the vanishing gradient problem.

During preprocessing for transformers, we added the time feature into our dataset. However, categorical features, such as time, cannot be fed to learning models before being converted into numerical values using Sklearn's one-hot [5] or Panda's get_dummies [4] encoding methods. When dealing with time series data, the data carries a timestamp which only provides the linear order of the points in ascending or descending. This phenomenon hinders patterns, i.e., it loses the connection between two consecutive data points. In our case, it hinders the cyclic pattern of hours of the day, days of the week, etc. For example, internet traffic usually gets congested when most users are active which is after-hours, lunch breaks, or weekends.

Therefore, to incorporate this cyclic pattern information into an interpretable feature, we utilized cosine and sine functions with cyclical behavior. These functions took the time data and transformed it into two features, giving each data point a unique value. Once our data was encoded, we fed four data sequences to the model to receive one output that indicated the appropriate *action* to take. The use of transformers and the inclusion of cyclical patterns in the time feature allowed the Stateless ARE to outperform the Stateful ARE in terms of accuracy and efficiency.

Chapter 5

Evaluation and Results

In Section 4.2, it was explained that the collected dataset has classes with uneven distribution of data points. This creates several issues. The first issue is that the AI algorithm becomes greedy and attempts to maximize its accuracy by favoring the predominant class. It does so by labeling most of the data with the label of the majority class to inflate the overall accuracy. This leads to biased results since the model is more interested in classifying the majority class accurately, and the minority class is ignored. The other issue is that the minority class is not competent enough to accurately describe the minority class features, and therefore, the model is unable to accurately describe the minority class features. These problems deteriorate the efficiency of the model since the model is biased towards the majority class, and we are more interested in classifying the minority class accurately.

To address these problems, we adopted ensemble algorithms as well as neural networks with weight balancing biases. Ensemble methods are used to unite weak models to form a more powerful algorithm whose predictive performance is boosted [11]. Ensemble models are efficient in classifying the minority classes in unbalanced datasets [22, 33]. Weight balancing, on the other hand, biases the model towards the class of interest by multiplying the loss of every data point by a constant value depending on their class. This approach results in an overall balance in the dataset [52].

The following subsections present the results of both *Stateful* and *Stateless* AREs. The Stateful ARE consists of two main modules; state classifier and action recommender. While the Stateless ARE consists of only one module, action recommender.

5.1 Stateful Action Recommendation Engine

5.1.1 State Classification

We split the dataset 60/40. 40% of the data is used for testing, and the remaining 60% of the data for training. We chose DT, GB, XGB, and a fully connected NN for the state classification task. The amount of training data was optimum for XGB model which returned the finest results. DT was too simple and NN was too complex. The volume of data was not enough to train the NN model efficiently. In Table 5.1, we listed the parameters of the four models. In Table 5.2 we presented the overall accuracies. As it is shown, all the models provided minimum of 97% accuracy.

Table 5.1: Models' parameters for State Classification.

DT	Split criterion: gini, Splitter: best, Depth:19, Leaves:149 Class Weight: 'Balanced'
GB	Learning rate: 0.1, Max depth: 3, No. of estimator: 100 Class Weight: 'Balanced'
XGB	Learning rate: 0.3, Max depth: 6, booster: gbtree, No. of estimator: 100, Class Weight: 'Balanced'
NN	Learning rate: 0.01, Hidden layers: 2, Neurons/Layer: 128, Optimizer: Adam

Table 5.2: Overall Model Accuracies for State Classification

Model	**DT**	**GB**	**XGB**	**NN**
Accuracy	97.1%	98.1%	99.0%	98.6%

We observed a similar phenomenon in precision and recall values provided in Tables 5.3 and 5.4 for state classification and fault localization. In comparison with the other models, XGB classified almost every class with higher proficiency. Precision values of the Normal class are between 0.97 and 0.99 for all the models. And for the NPI class, the value was 0.99 or 1. The demotion occurred in classifying Congestion classes. Precision values for this class were between 0.73 and 0.95. DT yielded the lowest value in identifying traffic congestion in PATH1 whereas XGB yielded the highest value in identifying traffic congestion in PATH2. Recall values is analogous with precision values. Normal class were again over 0.97 and NPI are between 0.98 and 1. Once again, we observe that the models struggled to classify congestion states. The lowest performance belongs to DT at 0.76

for Congest:P1 and highest belongs to XGB at 0.91 for Congest:P2 class. Although the performances for the congestion classes were mediocre, XGB still performed comparably better than other models with values over 0.90. Furthermore, when compared Congest:P1 and Congest:P2 classes, every model was more efficient in classifying Congest:P2. Hence, it is evident that among the models we have tested, XGB is the best model, DT is the poorest, GB and NN were very similar. We believe that NN didn't reach its potential due to lack of data.

Moreover, we noticed that the most common confusion models faced is identifying between the Normal and Congestion classes. After extensive experiments, we concluded that this confusion happens by cause of: (i) The class labeling policy expressed in Algorithm 4.1 labels the network state as congestion immediately after one of the paths is assigned more than 3 clients. (ii) Even though the class is labeled as congestion, it takes time for congestion effect to be seen clearly in the SLA and QoE metrics since the sent packets take time to suffer from high SLA (if not dropped because of congestion) due to the considerable size of the routers' buffer. That is why the data metrics show the congestion only after the buffer floods which takes time to be reflected. (iii) The last reason is related to the previous reason. It is due to the phenomenon called ML-model sampling frequency which is the frequency that the model takes decision. We know that it takes time for SLA and QoE metrics to reflect the congestion values in the data. When the model takes the decision before this reflection, it will classify the network state as Normal although we know that it is Congestion since we introduced the extra client to the network. This is the root of the confusion. However, when the model takes the decision after high SLA values, it will classify the state as Congestion and the ground truth label is also Congestion.

In Figure 5.1, we further compared the models' performances by using the normalized confusion matrices. In the figure, it shows that the models can easily detect NPI classes but the struggle happens between the normal and congestion classes. From the figure we can conclude that, the only considerable deterioration occurs between these two classes. In addition, DT and GB were unsuccessful in distinguishing Congest:P1 and Congest:P2. GB and NN yielded similar results in classifying Congest:P1. Although the other models failed to detect the congestion states, XGB was successful on this task.

Figure 5.2 depicts the ROC curves of the classifiers over discrimination thresholds. As shown, the models are capable of distinguishing Normal and NPI classes with AUC values of 0.80 and 1.00. However, we observe the same trend as seen in precision, recall, and confusion matrices. Once again, DT provided the lowest performance whereas GB and NN similar, while XGB provided the highest. Despite the fact that NN models produces state-of-the-art results in complex tasks such as object detection, face recognition, or multi class image classification, they are restrained with a critical barrier which is that their

Table 5.3: Precision for each class for State Classification

State / Model	Normal	NPI:3→5	NPI:2→4	NPI:2→1	NPI:3→2	NPI:1→3	Congest:P1	Congest:P2
DT	0.97	0.99	0.99	0.99	0.99	0.99	0.73	0.79
GB	0.98	1	0.99	0.99	1	1	0.87	0.86
XGB	0.99	1	1	0.99	0.99	1	0.93	0.95
NN	0.98	1	0.99	1	0.99	0.99	0.87	0.90

Table 5.4: Recall for each class for State Classification

State / Model	Normal	NPI:3→5	NPI:2→4	NPI:2→1	NPI:3→2	NPI:1→3	Congest:P1	Congest:P2
DT	0.97	1	0.98	0.99	1	0.99	0.76	0.78
GB	0.98	1	0.99	1	0.99	0.99	0.80	0.85
XGB	0.99	1	0.99	1	0.99	1	0.90	0.91
NN	0.98	1	0.99	0.99	0.99	1	0.86	0.89

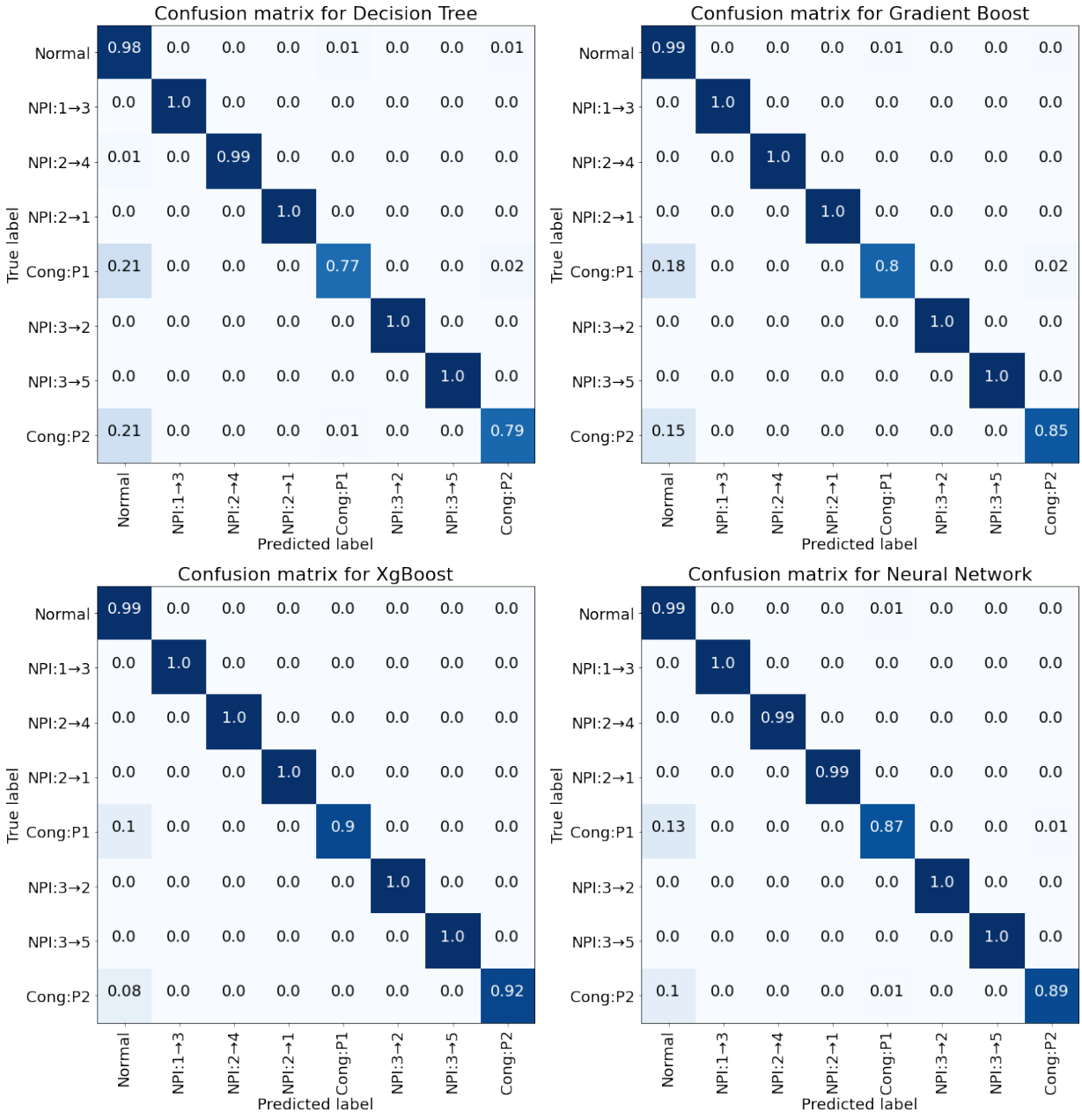


Figure 5.1: Confusion Matrices for DT, GB, XGB, and NN for State Classification.

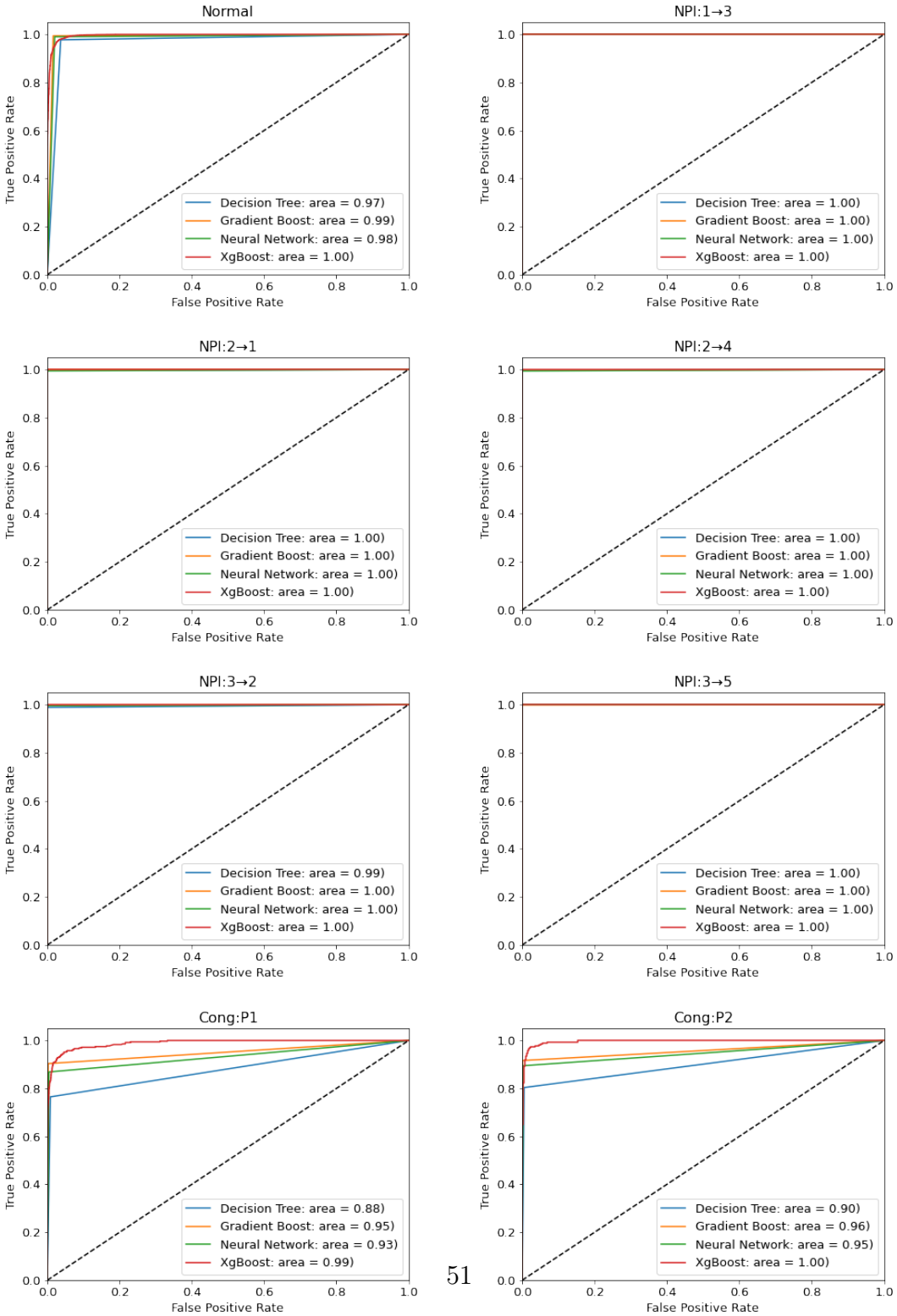


Figure 5.2: ROC Curves for DT, GB, XGB, and NN for state classification.

performance is limited by the amount of labeled data instances. The amount of available data is directly proportional with the performance measurement of the model.

5.1.2 Action Recommendation

We put 75% of the data in the training set and 25% in the test set. During training, we adopted 10-fold cross validation. We tested DT, GB, and XGB. Model parameters and top 3 feature importance values are shown in Table 5.5 and Table 5.6. Feature importance values are calculated using Sklearn's *feature_importances* function. It uses Mean Decrease Impurity to average each fold to find the top 3 most important features. DT, GB, and XGB classification models returned 99.48%, 99.70%, and 99.87%, respectively. These accuracies are relatively high, however they are misleading. Since around 68% of the data belong to Do Nothing class shown in Figure 4.4(a) and 32% belong to the remaining classes shown in Figure 4.4(b).

Table 5.5: Models' parameters: all are hyper-parameters unless stated otherwise.

DT	Split criterion: gini, Splitter: best, Max depth:19, Class Weight: 'Balanced', Leaves: 149 (learned)
GB	Learning rate: 0.1, Max depth: 3, No. of estimator: 100 Class Weight: 'Balanced'
XGB	Learning rate: 0.3, Max depth: 6, booster: gbtree, No. of estimator: 100, Class Weight: 'Balanced'

In order to further inspect the model performances, we breakdown the remaining classes and analyze the F1 score for each of them, see Table 5.7. It can be seen that the Do Nothing action has the perfect score, 1. This leads us to the definite conclusion that every model can precisely detect the Do Nothing class. Models also perform well in detecting link issues

Table 5.6: Top-3 important features

DT	GB	XGB
Path1_RTT_Min: 29%	State_IsNormal: 28%	Path1_Jitter_Avg: 35%
Path1_Jitter_Avg: 9%	Path2_RTT_Min: 4%	State_IsNormal: 19%
State_IsNormal: 3%	Path1_PacketLoss: 7%	Router_Port0_Drop: 2%

with 0.97-1.00 F1 scores. On the other hand, models significantly deteriorate when faced to Congestion related action classes. The lowest performances belong to the TE P1 → P2, TE P1 → P3, and TE P2 → P3 rerouting actions. The lowest F1 scores are from TE P2 → P1 with GB having 0.84 and TE P3 → P1 with DT 0.81 and GB 0.88. It is essential to observe that these two classes are the ones with the least number of data points.

Table 5.7: Models F1-Score.

Model	**TE-Action**					
	P1-P2	P3-P2	P1-P3	P2-P3	P2-1	P3-1
DT	0.88	0.94	0.9	0.93	1	0.81
GB	0.910	0.98	0.93	0.92	0.84	0.88
XGB	0.92	1	0.96	0.97	1	0.96
Model	**Fix-Action**					
	DoNoth	**LR-1-3**	**LR-2-4**	**LR-2-1**	**LR-3-2**	**LR-1-3**
DT	1	0.99	0.99	0.97	1	1
GB	1	1	0.99	1	1	1
XGB	1	1	1	1	1	1

Furthermore, DT ranks as the worst efficient model. The only exception where it performed better than the other models is when rerouting actions TE P2 → P3 for congestion and TE P2 → P1 due to cost optimization policy. In these two instances, for the congestion action GB yielded slightly lower than DT. However, for the cost optimization action, GB's score is 0.16 lower. Nonetheless, XGB and the DL model are the best two models. Their results were either the same or one was imperceptibly better. This trend is carried out with the cost optimization policy actions in which DT and GB encountered hardship.

On the other hand, confusion matrices displayed minor differences from the F1 scores, see Figure 5.3. In comparison with DT and GB, XGB was still the most efficient model. Though, its scores in congestion action TE P1 → TE P2 and cost optimization action TE P3 → P1 were lower then GB and DT, respectively. In addition, all the models almost always mislabeled the actions as Do Nothing. Moreover, DT and GB followed similar outcomes.

In summary, the boosting algorithms (or boosting based models) performed the best results. Boosting improves the accuracy of weak learning algorithms, i.e., boosting can overcome the problem of underfitting [68] which was a significant challenge while we were training the models. Underfitting occurred because the learning algorithm failed to fit the

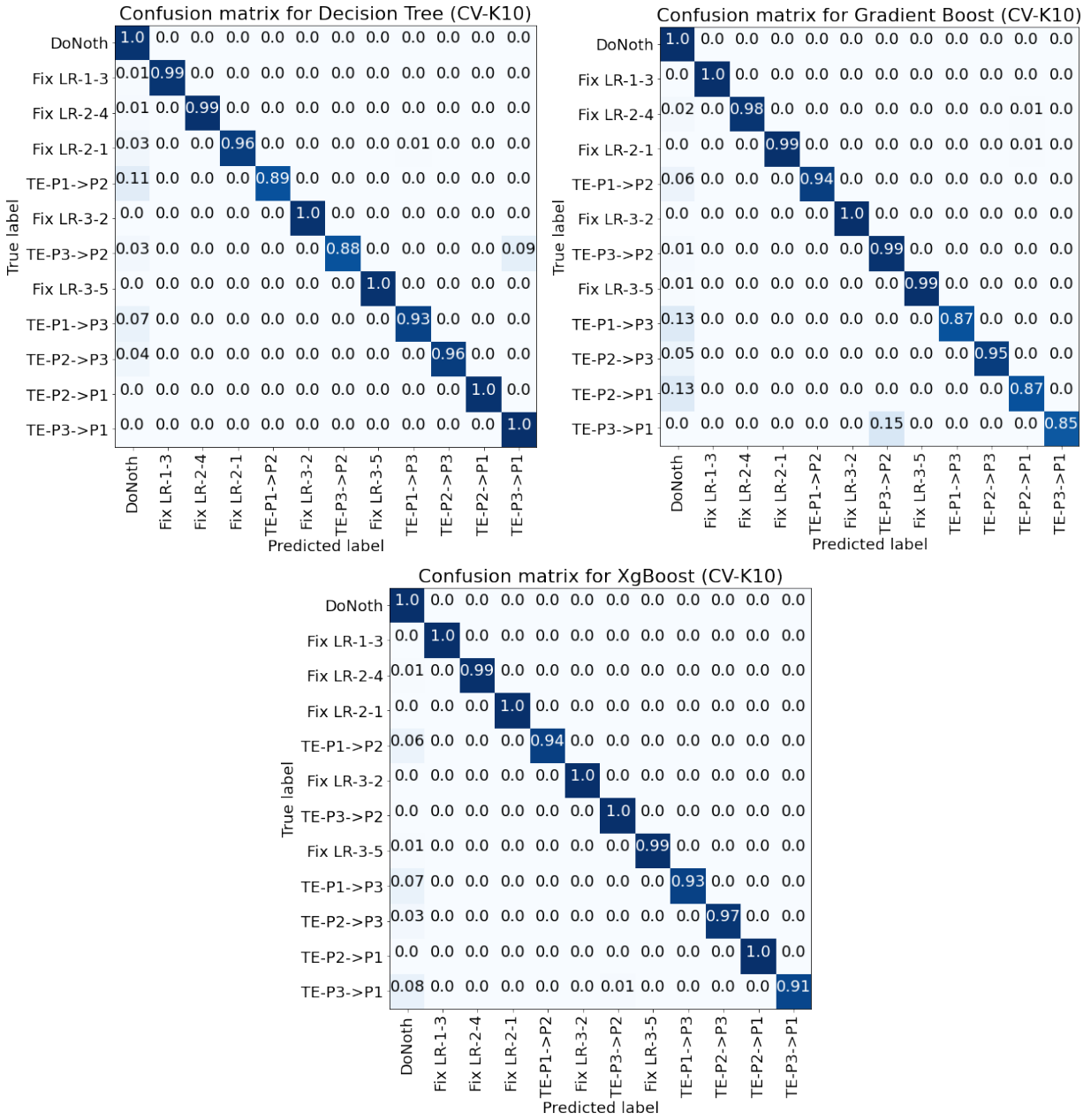


Figure 5.3: Confusion Matrices for DT, GB, and XGB for Action Recommendation.

training data well enough. Boosting reweights the training examples iteratively and trains weak classifiers on this newly generated reweighted data. Therefore the resulting boosted classifier can have a much lower error rate than any of the weak classifiers on their own, even when the weak classifiers underfit the data. Our results endorses this concept.

5.2 Stateless Action Recommendation Engine

5.2.1 Feed Forward Neural Network

In order to test the performance of the stateless ARE, we planned two different approaches; ML and DL. For the ML model, initially, we started testing the XGB algorithm that performed well with stateful ARE. However, its performance was insufficient in this testbed. Hence, we developed the FFNN model, see Figure 5.4. It includes 132 inputs, 3 fully connected hidden layers with 264, 128, and 80 neurons, and finally 17 possible outputs. The FFNN model is designed to generate more problems during training phase which means that, even though, our system design has 12 actions to take on the network to restore it, see Table 4.5, we introduced 5 additional actions which breaks or congests certain links to create more data.

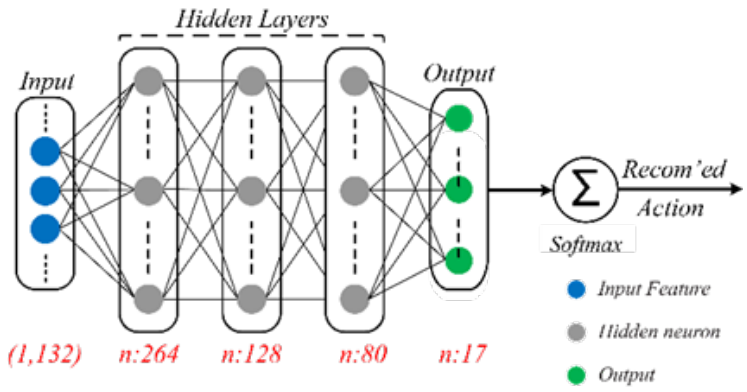


Figure 5.4: Layers of FFNN.

To better address the non-linearity of our data, Leaky ReLU activation function is selected and implemented between input and hidden layers. In the output layer, SoftMax activation function is used to select the output with the highest probability. Table 5.8 lists the FFNN parameter choices for the training process.

Table 5.8: FFNN's parameters.

Parameter	Value
Train:Test	70:30
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Crossentropy
Number of Epochs	20
Batch Size	1000

We applied the stateless ARE on the same topology shown in Figure 4.1 with the same expert rules defined for stateful ARE. We employed 70%:30% split between training and testing sets. During training, we adopted 10-fold cross validation. Figure 5.5 shows the confusion matrix of FFNN model on stateless ARE. It can be inferred that the overall performance of stateless ARE is satisfactory with the exception of 9% classification errors in 3 classes; TE-P1->P2, TE-P3->P2, and TE-P1->P2. These classes are generally mistaken with the Do Nothing class.

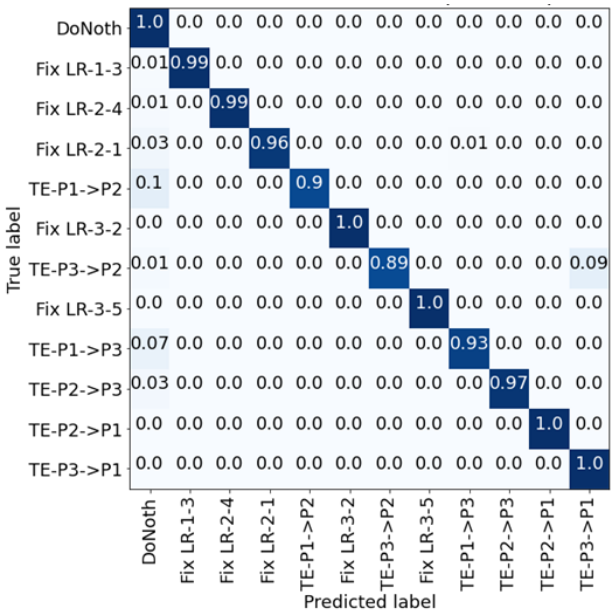


Figure 5.5: Confusion matrix of FFNN model on Stateless ARE.

5.2.2 Transformers

The dataset was split into two parts, 70% of the data was used for training, and the remaining 30% was used for testing. he transformer architecture used in this study followed the conventional design shown in Figure 2.11. It consisted of one embedding layer, one transformer encoder, one decoder layer, and four hidden layers. The learning rate was set to 0.001, the epoch was 25, and the batch size was 200, as detailed in Table 5.9. Unlike the FFNN model, the transformer model was fed with four data sequences to receive one action output. In addition to the existing features of the dataset, we introduced a transformed version of the time stamps using sine and cosine functions to incorporate cyclical time data. When dealing with time series data, the timestamp only provides the linear order of the points in ascending or descending. This phenomenon hinders patterns and loses the connection between two consecutive data points, hindering the cyclic pattern of hours of the day, days of the week, etc.

As shown in Figure 5.6, the transformer model improved the results in various actions, such as TE-P2->P3, TE-P1->P3, TE-P3->P2, TE-P1->P2, Fix LR-2-1, Fix LR-2-4, Fix LR-1-3, while slightly decreasing the results for Fix LR-3-5 and DoNothing actions. It enhanced the results Fix LR-2-1 (0.04%), TE-P1->P2 (0.05%), and TE-P3->P2 (0.08%) while the rest is around 0.01% increase. These results indicate that the transformer model outperformed the previous models, and its performance can be further improved by incorporating more relevant features into the dataset.

Table 5.9: Transformer's parameters.

Parameter	**Value**
Train:Test	70:30
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Crossentropy
Number of Epochs	25
Batch Size	200

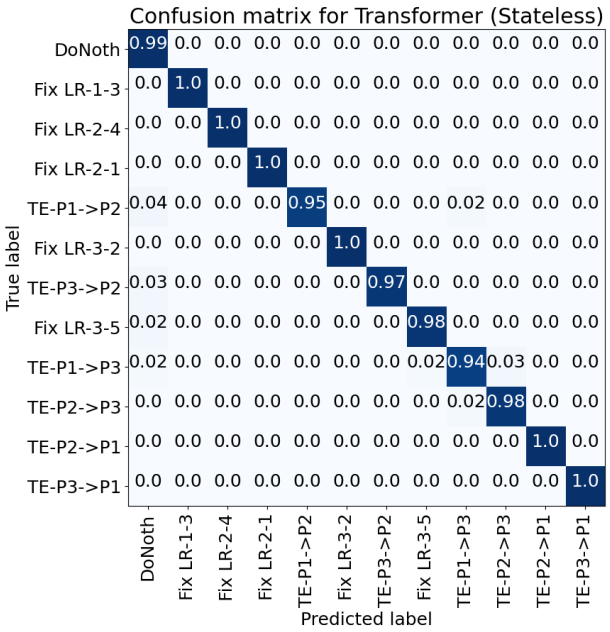


Figure 5.6: Confusion matrix of Transformer model on Stateless ARE.

5.3 Real World Experiment

5.3.1 Feed Forward Neural Network Model

In order to further evaluate the effectiveness of our work and better understand the impact of misclassifications in the system, we conducted a live emulation that simulates real-life scenarios that NOCs encounter. The primary metric we used for comparison was the QoE-OPEX. In the experiment, we compared four scenarios:

- Static Network: uninterrupted network, i.e., a network without any intervention from a person or an algorithm
- NOC-Mimic: OPEX aware expert rules
- Stateful ARE: out of the stateful ARE models, we opted the best model which is XGB
- Stateless ARE

Based on the results, we chose the XGB algorithm to represent the stateful ARE since it outperformed the other models, as discussed in Section 5.1.1. Figure 5.8 depicts the closed-loop architecture that we used for the live comparison of the different scenarios. We observed two key situations.

Firstly, the stateful ARE still performed better than the stateless ARE, but it closely followed the NOC-Mimic curve. This suggests that even though the network state detector makes occasional errors, it still improves overall performance. This conclusion is supported by the confusion matrices shown in Figure 5.1 (for the XGB model) and Figure 5.5, which demonstrate that the network state detector provides higher accuracy than the stateless ARE. Secondly, the stateless ARE outperformed the static network, despite performing less effectively than the stateful ARE. This indicates that the FFNN model on the stateless ARE was able to learn the NOC rules and suggests that there is potential for further improvement in its performance.

5.3.2 Transformers

In our previous work, we conducted a live emulation experiment to compare the QoE-OPEX metric of Static Network, NOC-Mimic, Stateful ARE, and Stateless ARE models in real-life situations. Our findings indicated that while the stateless ARE DL model did not surpass the performance of the Stateful ARE, it demonstrated superior outcomes compared to the static network. However, we aimed to further improve the performance of the stateless ARE model by incorporating transformers into our approach.

Transformers are a type of neural network architecture that has gained significant popularity in NLP tasks. However, their potential to learn and extract meaningful insights from sequential data made them suitable for our problem since we deal with sequential time series data. We hypothesized that by incorporating transformers into our stateless ARE model, we could improve its performance in network anomaly detection and network automation.

Initially, the transformer model took some time to yield better results. After around 500 iterations, it outperformed the XGB and FFNN models, as shown in Figure 5.8. This improvement in performance can be attributed to the transformer's ability to learn and effectively utilize the cyclic behavior inherent in the data. By transforming the time stamps of our data into sine and cosine functions, we were able to represent them as cyclical features that the transformer could learn from. This enabled the model to extract meaningful insights from the pattern, thus surpassing other models.

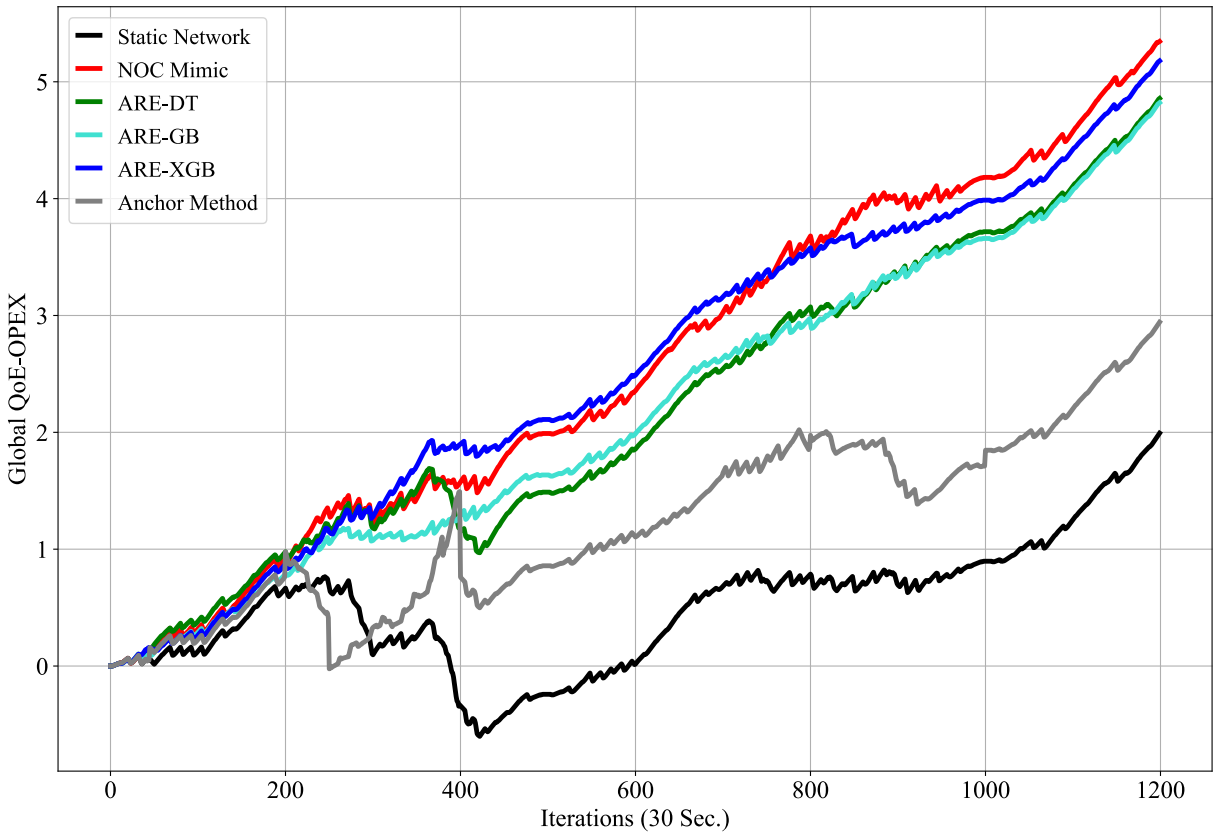


Figure 5.7: Cumulative QoE-OPEX comparison between static network, NOC rules, stateful ARE models, and Anchor method.

The contributions of the transformer model to our work are significant. The transformer model improved the performance of the stateless ARE model in detecting network anomalies. This is a critical task in NOCs, as early detection and mitigation of network anomalies can prevent significant downtime and customer dissatisfaction due to degradation in QoS and QoE. By improving the performance of the stateless ARE model, we have demonstrated the potential benefits of our approach in real-world scenarios.

The transformer model has also highlighted the importance of considering the cyclic behavior of time stamps in sequential data. This is particularly relevant in network operations, where network traffic and behavior are cyclical in nature. For instance, the network traffic usually gets congested after work hours and during weekend. By incorporating cyclical features into our transformer model, we were able to capture the inherent pat-

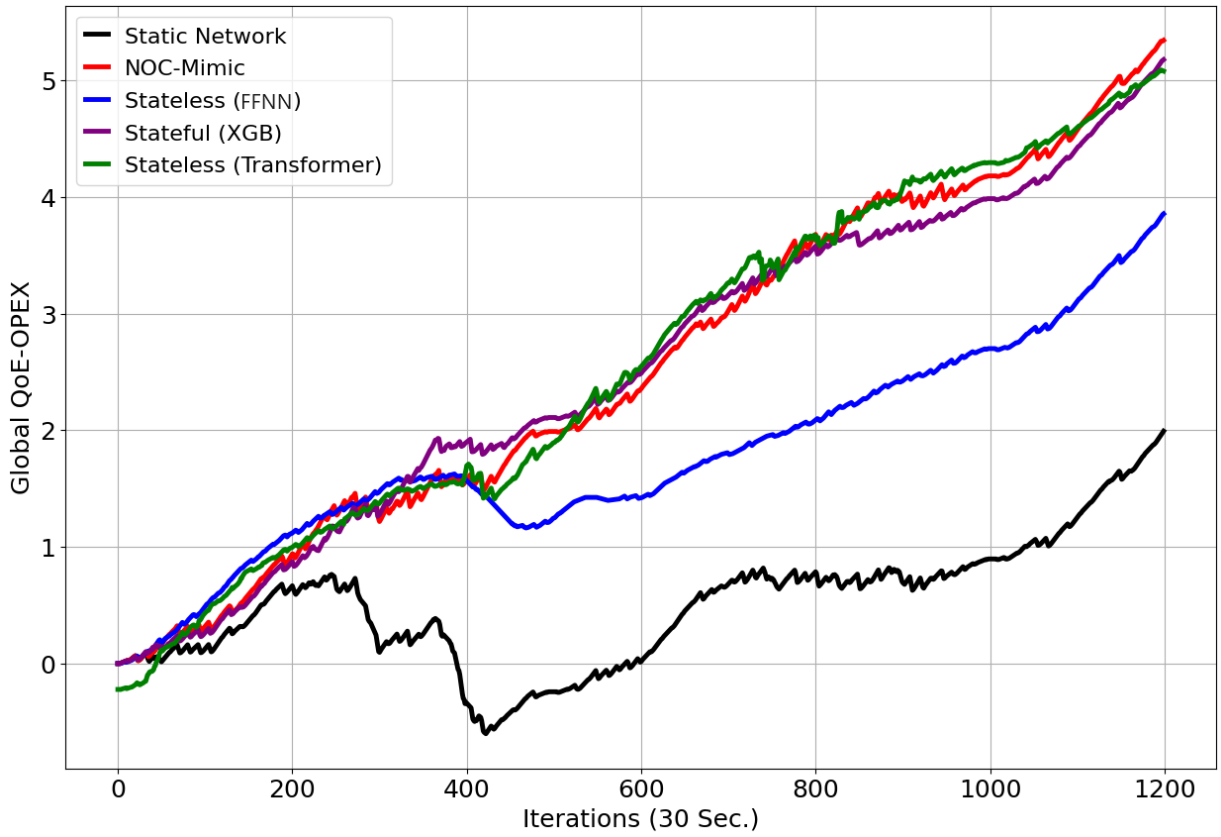


Figure 5.8: Cumulative QoE-OPEX comparison between static network, NOC rules, XGB model for stateful ARE, DL (FFNN) and Transformer model for stateless ARE.

terns in our dataset and therefore, improve the accuracy of our model in detecting network anomalies.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In recent years, ML techniques have gained considerable attention in various fields, including network operations. We followed this trend and applied ML in network operations and developed one of the most promising applications Action Recommendation Engine (ARE). ARE automates the manual tasks performed by NOC technicians and recommend the best available action to maintain a stable network while also accommodating OPEX requirements.

In this section, we will summarize our research on developing stateful and stateless ARE models and their real-life applications. We will also summarize the state-of-the-art transformer models and how they have advanced stateless ARE. Finally, we will conclude by highlighting the potential of combining transformer models with stateless ARE models in other fields beyond network operations.

Our research on developing stateful ARE model aimed to incorporate previous network states and actions into the decision-making process. The stateful ARE model can accurately detect the network state and make more informed decisions based on previous network states and actions. We used an ML model to classify the network status into three categories: *congestion*, *network physical issue*, and *normal*. To collect data for our model, we designed the testbed in GNS3 and collected the necessary information from this platform. Subsequently, we applied three different models to the dataset and found that XGB model exhibited the most superior performance, achieving an accuracy of network state classification of 99%. The outputs from the model, which denote the predicted network

states, are merged with the original dataset. This new augmented dataset is then fed into a distinct ML-based model which generates a series of actions, such as *do nothing*, *reroute*, and *fix issues*, based on the predicted network states. Consequently, we have successfully developed the first ML-based action recommendation engine for NOCs.

The live emulation experiment conducted in this study provided valuable insights into the practical implications of the proposed Stateful and Stateless ARE models. The experiment aimed to compare the performance of different approaches, including the Static Network, NOC-Mimic, Stateful ARE, and Stateless ARE, in terms of their impact on (QoE)-OPEX. The results of the experiment showed that the Stateful ARE model, specifically the XGB algorithm, outperformed the stateless ARE model, although the latter still outperformed the Static Network. This suggests that the Stateful ARE model is more effective in accurately detecting the network state, and its ability to incorporate previous network states and actions enables it to make more informed decisions.

Our research on Stateful ARE models has practical implications for NOCs, where the ability to quickly and accurately diagnose and resolve network issues can have a significant impact on overall network performance and user satisfaction. ARE can automate the manual tasks performed by NOC technicians and recommend the best available action to maintain a stable network while also accommodating OPEX requirements.

The Stateful ARE model is particularly well-suited for complex network scenarios, where previous network states and actions can provide valuable insights into the current network state and the appropriate action to take. In such scenarios, the Stateful ARE model can help NOC technicians make more informed decisions and improve the overall network performance and user satisfaction.

While Stateful outperformed or closely followed the performance of the NOC technicians, and therefore presented a satisfactory performance, it suffered from a phenomenon called error propagation. This was due to the fact that it consists of two cascaded ML models, which meant that any errors that occurred during the state classification phase would propagate into the secondary phase. We then proceeded to explore the possibility of implementing the ARE without state measurement. Our investigation of the possibility of implementing the ARE without state measurement led to the development of stateless ARE models.

Stateless ARE models were designed to recommend actions based on its input features without the need to incorporate previous network states and actions. We developed the FFNN model to represent the Stateless ARE during our experiments. Initially, the FFNN model did not exceed the performance of the original Stateful ARE model, but it demonstrated promising results by outperforming the static network. Furthermore, the

Stateless ARE model has several advantages over the Stateful ARE. For example, it does not require storing past network states and actions, which reduces the computational and memory requirements. Additionally, the Stateless ARE model is more flexible and can adapt to dynamic network conditions. This flexibility enables the model to recommend appropriate actions in real-time even in the case when networks expand.

However, the FFNN model used in the Stateless ARE had a limited capacity to represent complex relationships among features. Combining this conclusion with the insight we gained during the performance comparison of Stateful and Stateless ARE, we decided to investigate applying a new DL architecture into our research problem, *transformers*. The transformer architecture is designed to handle such complex relationships efficiently. Therefore, we hypothesized that incorporating transformers into the stateless ARE would improve its performance.

Our implementation of the stateless ARE with transformers architecture demonstrated significant improvements in performance compared to the initial stateless ARE model and outperformed the Stateful ARE model. The transformer-based Stateless ARE achieved a classification accuracy of 94-99% in minority classes, which is a remarkable improvement over the FFNN-based Stateful ARE's 89-97% accuracy and XGB-based Stateful ARE's 93-97% accuracy. These results highlight the importance of selecting an appropriate DL architecture that can capture the complex relationships among the input features and produce accurate results in action recommendation task.

Hence, we can conclude that one of the significant contributions of our work is the implementation of transformer-based architectures into the Stateless ARE model, which resulted in improved performance in predicting the appropriate actions from raw data. Transformers have gained considerable attention in recent years, and their application in NLP has been particularly noteworthy. Our work demonstrates their potential for improving the performance of action recommendation engine in automating network operations.

The transformer architecture's ability to model long-term dependencies and relationships between sequences makes it well-suited for handling time-series data like the one used in our network operation use case. By feeding multiple data sequences, in our case four sequences, to receive a single action output, we were able to train the transformer model to effectively capture the temporal dependencies in the input data, resulting in better performance in predicting the appropriate action to take.

Our findings suggest that incorporating transformers in the stateless ARE model can greatly improve the accuracy and efficiency of network operations. This has significant practical implications for NOCs, where the ability to swiftly and accurately diagnose and resolve network issues can have a significant impact on overall network performance and

user satisfaction.

In addition, our work also highlights the potential of ensemble algorithms and weight balancing biases in addressing the issue of imbalanced data in network operations. The results showed that these techniques can effectively improve the accuracy of our models in classifying minority classes, which is critical for accurately diagnosing and resolving network issues when dealing with network anomalies.

Overall, our work contributes to the research on ARE in network operations and demonstrates the potential of state-of-the-art ML techniques such as transformers in improving their performance.

In conclusion, our work has demonstrated the potential benefits of combining transformers with stateless ARE models for network anomaly detection. The transformer model improved the performance of the Stateless ARE model in detecting network anomalies, while also highlighting the importance of considering the cyclic behavior of time stamps in sequential data. Our approach has great potential in automating network operations and satisfying OPEX rewuirements, and we look forward to further exploring its applications.

6.2 Future Work

Although the current system operates on high performance, it leaves important venues for further research; *scalability*. Scalability issue stems from the following challenges: network architecture and machine learning model architecture.

- *Network architecture.* Currently, the system requires adjustment whenever an expansion occurs in the ISP's network. To mitigate this issue, Graph Neural Network (GNN) models could be explored. They are specialized to work on graph/network data and should scale with the expansion of the graph representation.
- *Dataset scalability.* As network topology expands, dataset features including the number of states and actions increase. Hence the current model becomes obsolete, the current models need to be trained from scratch. This scalability issue could be addressed by GNNs since they use graphs to represent the data and could scale and adapt to any changes in the network topology, therefore the dataset.
- *MLOPs (Lab to Production).* As with all AI models, ARE is prone to operating in a real-world setting where the data may differ slightly from the data it was trained on. To address this issue, we intend to utilize the ML-Ops workflow to deploy ARE

in a real-world context and automatically implement transfer learning or retraining as required, should there be any observation in performance decrease.

- *Explainable AI.*: We did not delve into the field of explainable AI for our project since our system is not classified as human life-critical such as medical, military, aviation systems, or autonomous vehicles. However, providing an explanation for the model's decision-making process may be a valuable input to ARE and therefore is considered for future work.

- *Transformers.*: While developing a powerful stateless ARE, the first FFNN model did not outperform its stateful equivalent, however, it outperformed the static network which encouraged us to apply a new architecture to our problem. Here where the transformer model came into the picture. The experimental findings have clearly demonstrated the efficacy of the transformer model. Nonetheless, the utilization of transformers in time series problems is an active area of research. Our model could benefit from further optimizations.

- *Reinforcement Learning (RL).*: The success of transformers has led to the hypothesis that the stateless action recommendation engine could outperform expert rules significantly, with further optimization or by employing new AI methods such as reinforcement learning (RL). Consequently, we have been motivated to explore and incorporate novel DL architectures, including RL, into our research. RL has demonstrated remarkable proficiency in complex tasks after undergoing extensive training and data processing. As such, we anticipate that RL can serve as a valuable asset to tackle our problem.

- *Graph Neural Networks (GNNs).*

- *Less Data.*: Finding a way to combine unsupervised learning with supervised learning to teach NNs how to learn with less data is a promising area of research. Furthermore, teaching NNs to accumulate their knowledge will make them more effective and efficient in learning new things, and thus, fewer data will be required for training.

References

[1] Cisco mpls. https://www.cisco.com/c/en/us/products/ios-nx-os-software/multiprotocol-label-switching-mpls/index.html.

[2] Gns3. https://www.gns3.com/. Accessed: 2022-09-22.

[3] Juniper mpls. https://www.juniper.net/documentation/en_US/junos\/topics/concept/mpls-security-overview.html.

[4] pandas.get_dummies. https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html/. Accessed: 2023-03-10.

[5] sklearn.preprocessing.onehotencoder. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html/. Accessed: 2023-03-10.

[6] Statista. https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/. Accessed: 2022-10-28.

[7] Uptime institute. https://uptimeinstitute.com/. Accessed: 2022-10-31.

[8] Wireshark, go deep. https://www.wireshark.org/. Accessed: 2022-10-28.

[9] Maryam Amiri, Hussein Al Osman, and Shervin Shirmohammadi. "Game-aware and SDN-assisted bandwidth allocation for data center networks". *2018 IEEE conference on multimedia information processing and retrieval (MIPR)*, pages 86–91, 2018.

[10] Ahmed Amokrane, Rami Langar, Raouf Boutabayz, and Guy Pujolle. "Online flow-based energy efficient management in wireless mesh networks". *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 329–335, 2013.

[11] Mourad Azhari, Altaf Alaoui, Abdallah Abarda, Badia Ettaki, and Jamal Zerouaoui. "Using ensemble methods to solve the problem of pulsar search". *Springer International Conference on Big Data and Networks Technologies*, pages 183–189, 2019.

[12] Abdullah Baz. "Bayesian machine learning algorithm for flow prediction in sdn switches". *IEEE 1st International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–7, 2018.

[13] Rick Blum and J Kaplan. "Network operations centers". *International Network Services Network Industry Survey (March)*, 2004.

[14] Andriy Burkov. *The hundred-page machine learning book*, volume 1. Andriy Burkov, Quebec City, QC, Canada, 2019.

[15] Guo Chen, Yuanwei Lu, Yuan Meng, Bojie Li, Kun Tan, Dan Pei, Peng Cheng, Layong Luo, Yongqiang Xiong, Xiaoliang Wang, and Youjian Zhao. "Fuso: Fast multi-path loss recovery for data center networks". *IEEE/ACM Transactions on Networking*, 26(3):1376–1389, 2018.

[16] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[17] Iyad Lahsen Cherif and Abdesselem Kortebi. "On using extreme gradient boosting (xgboost) machine learning algorithm for home network traffic classification". *2019 IEEE Wireless Days (WD)*, pages 1–6, 2019.

[18] Radostin Cholakov and Todor Kolev. "Transformers predicting the future. applying attention in next-frame and time series forecasting". *arXiv preprint arXiv:2108.08224*, 2021.

[19] David Côté. "Using machine learning in communication networks". *Journal of Optical Communications and Networking*, 10(10):D100–D109, 2018.

[20] Qian Dong, Jun Li, Yuxiang Ma, and Shujun Han. "A path allocation method based on source routing in sdn traffic engineering". *2019 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 163–168, 2019.

[21] Jerome H Friedman. "Stochastic gradient boosting". *2002 Elsevier Computational statistics & data analysis*, 38(4):367–378, 2002.

[22] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches". *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2011.

[23] Fabien Geyer and Georg Carle. "Learning and generating distributed routing protocols using graph-based deep learning". In *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pages 40–45, 2018.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[25] Lav Gupta, Tara Salman, Maede Zolanvari, Aiman Erbad, and Raj Jain. "Fault and performance management in multi-cloud virtual network services using ai: A tutorial and a case study". *Elsevier, Computer Networks*, 165:106950, 2019.

[26] William L. Hamilton. "Graph representation learning". *Synthesis Lectures on Artifical Intelligence and Machine Learning, Morgan & Claypool Publishers*, 14(3):1–159, 2020.

[27] Guangjie Han, Yuhui Dong, Hui Guo, Lei Shu, and Dapeng Wu. "Cross-layer optimized routing in wireless sensor networks with duty cycle and energy harvesting". *Wiley Online Library, Wireless communications and mobile computing*, 15(16):1957–1981, 2015.

[28] Yuxiu Hua, Zhifeng Zhao, Zhiming Liu, Xianfu Chen, Rongpeng Li, and Honggang Zhang. "Traffic prediction based on random connectivity in deep learning with long short-term memory". pages 1–6, 2018.

[29] Auwal Sani Iliyasu and Huifang Deng. "Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks". *IEEE Access*, 8:118–126, 2019.

[30] Shashwat Jain, Manish Khandelwal, Ashutosh Katkar, and Joseph Nygate. "Applying big data technologies to manage qos in an sdn". *12th International Conference on Network and Service Management (CNSM), IEEE*, pages 302–306, 2016.

[31] Yong Jin, Masahiko Tomoishi, and Satoshi Matsuura. "Detection of hijacked authoritative dns servers by name resolution traffic classification". *2019 IEEE International Conference on Big Data*, pages 6084–6085, 2019.

[32] Renuga Kanagavelu and Yongqing Zhu. "A pro-active and adaptive mechanism for fast failure recovery in sdn data centers". *Future of Information and Communication Conference, Springer*, pages 239–257, 2018.

[33] Taghi M Khoshgoftaar, Alireza Fazelpour, David J Dittman, and Amri Napolitano. "Ensemble vs. data sampling: Which option is best suited to improve classification performance of imbalanced bioinformatics data?". pages 705–712, 2015.

[34] Rohit Kumar, U Venkanna, and Vivek Tiwari. A binary classification approach for time granular traffic in sdwmn based iot networks. *2020 International Conference on Communication Systems & Networks (COMSNETS), IEEE*, pages 531–534, 2020.

[35] Subham Kumar, Gaurang Bansal, and Virendra Singh Shekhawat. "A machine learning approach for traffic flow provisioning in software defined networks". *2020 International Conference on Information Networking (ICOIN), IEEE*, pages 602–607, 2020.

[36] Dan Li, Yunfei Shang, and Congjie Chen. "Software defined green data center network with exclusive routing". *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1743–1751, 2014.

[37] Jinxia Li, Shiliang Yu, Yong Zhang, Jun Hu, Yijun Jiang, Lianjie Cui, Hongbo Zhang, and Xinxin Zhou. "A transformer-based framework for predicting adverse events in pediatric patients". *Journal of Healthcare Engineering, Nature*, pages 1–13, 2021.

[38] Qiliang Li, Jie Cui, Hong Zhong, Yichao Du, Yonglong Luo, and Lu Liu. "LBBESA: An efficient software-defined networking load-balancing scheme based on elevator scheduling algorithm". *Concurrency and Computation: Practice and Experience, Wiley Online Library*, 32(16):e5222, 2020.

[39] Qing Li, Yang Liu, Zhijie Zhu, Hengtong Li, and Yong Jiang. "BOND: Flexible failure recovery in software defined networks". *Computer Networks, Elsevier*, 149:1–12, 2019.

[40] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. "Temporal fusion transformers for interpretable multi-horizon time series forecasting". *International Journal of Forecasting, Elsevier*, 37(4):1748–1764, 2021.

[41] Yang Liu, Cheng Lyu, Xin Liu, and Zhiyuan Liu. "Automatic feature engineering for bus passenger flow prediction based on modular convolutional neural network". *IEEE Transactions on Intelligent Transportation Systems*, 22(4):2349–2358, 2020.

[42] Yong Liu, Huaxi Gu, Kun Wang, Xiaoshan Yu, and Yunhao Wang. "An adaptive failure recovery mechanism based on asymmetric routing for data center networks". *The Journal of Supercomputing, Springer*, 77(2):2103–2123, 2021.

[43] Nadeem Malibari, Iyad Katib, and Rashid Mehmood. "Predicting stock closing prices in emerging markets with transformer neural networks: The saudi stock exchange case". *International Journal of Advanced Computer Science and Applications, Science and Information (SAI) Organization Limited*, 12(12), 2021.

[44] Ali Malik, Benjamin Aziz, Mo Adda, and Chih-Heng Ke. "Smart routing: Towards proactive fault handling of software-defined networks". *Computer Networks, Elsevier*, 170:107104, 2020.

[45] Bomin Mao, Zubair Md Fadlullah, Fengxiao Tang, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. "Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning". *IEEE Transactions on Computers*, 66(11):1946–1960, 2017.

[46] Dirk Merkel. "Docker: lightweight linux containers for consistent development and deployment". *Linux J*, 239(2):2, 2014.

[47] Albert Mestres, Eduard Alarcón, Yusheng Ji, and Albert Cabellos-Aparicio. "Understanding the modeling of computer network delays using neural networks". In *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks, ACM*, pages 46–52, 2018.

[48] Ayşe Rumeysa Mohammed, Shady A Mohammed, David Côté, and Shervin Shirmohammadi. "Machine learning-based network status detection and fault localization". *IEEE Transactions on Instrumentation and Measurement*, 70:1–10, 2021.

[49] Ayşe Rumeysa Mohammed, Shady A Mohammed, David Côté, and Shervin Shirmohammadi. "Stateless ARE: Action recommendation engine without network state measurement". *2022 IEEE International Symposium on Measurements & Networking (M&N)*, pages 1–6, 2022.

[50] Ayşe Rumeysa Mohammed, Shady A Mohammed, and Shervin Shirmohammadi. "Machine learning and deep learning based traffic classification and prediction in software defined networking". *2019 IEEE International Symposium on Measurements & Networking (M&N)*, pages 1–6, 2019.

[51] Shady A Mohammed, Ayşe Rumeysa Mohammed, David Côté, and Shervin Shirmohammadi. "A machine-learning-based action recommender for network operation centers". *IEEE Transactions on Network and Service Management*, 18(3):2702–2713, 2021.

[52] Ajinkya More. "Survey of resampling techniques for improving classification performance in unbalanced datasets". *arXiv preprint arXiv:1608.06048*, 2016.

[53] Tashreef Muhammad, Anika Bintee Aftab, Muhammad Ibrahim, Md Mainul Ahsan, Maishameem Meherin Muhu, Shahidul Islam Khan, and Mohammad Shafiul Alam. "Transformer-based deep learning model for stock price prediction: A case study on Bangladesh stock market". *International Journal of Computational Intelligence and Applications, World Scientific*, page 16, 2023.

[54] Sheraz Naseer, Yasir Saleem, Shehzad Khalid, Muhammad Khawar Bashir, Jihun Han, Muhammad Munwar Iqbal, and Kijun Han. "Enhanced network anomaly detection based on deep neural networks". *IEEE access*, 6:48231–48246, 2018.

[55] Nidhi Nidhi and D. K. Lobiyal. "Traffic flow prediction using support vector regression". *International Journal of Information Technology, Springer*, 14(2):619–626, 2022.

[56] Talha Ongun, Timothy Sakharaov, Simona Boboila, Alina Oprea, and Tina Eliassi-Rad. "On designing machine learning models for malicious network traffic classification". *arXiv preprint arXiv:1907.04846*, 2019.

[57] Jaehwa Park and JunSeong Kim. "A classification of network traffic status for various scale networks". *The International Conference on Information Networking 2013 (ICOIN), IEEE*, pages 595–599, 2013.

[58] Sanjeev Patel, Akash Gupta, Suman Kumari, Manjeet Singh, Vishnu Sharma, et al. "Network traffic classification analysis using machine learning algorithms". *2018 IEEE International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 1182–1187, 2018.

[59] Yuhuai Peng, Xiaoxue Gong, Lei Guo, and Dezhi Kong. "A survivability routing mechanism in SDN enabled wireless mesh networks: Design and evaluation". *China Communications, IEEE*, 13(7):32–38, 2016.

[60] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. "3D graph neural networks for RGBD semantic segmentation". In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5199–5208, 2017.

[61] Alyssa Quek, Zhiyong Wang, Jian Zhang, and Dagan Feng. "Structural image classification with graph neural networks". *2011 International Conference on Digital Image Computing: Techniques and Applications, IEEE*, pages 416–421, 2011.

[62] J. Ross Quinlan. "Induction of decision trees". *Machine Learning, Springer*, 1(1):81–106, 1986.

[63] Adam Rupe, Karthik Kashinath, Nalini Kumar, Victor Lee, James P Crutchfield, et al. "Towards unsupervised segmentation of extreme weather events". *arXiv preprint arXiv:1909.07520*, 2019.

[64] Gokay Saldamli, Himanshu Mishra, Naveen Ravi, Rahul Rao Kodati, Sahithi A Kuntamukkala, and Loai Tawalbeh. "Improving link failure recovery and congestion control in SDNs". *10th International Conference on Information and Communication Systems (ICICS), IEEE*, pages 30–35, 2019.

[65] Avdhoot Saple and Levent Yilmaz. "Agent-based simulation study of behavioral anticipation: anticipatory fault management in computer networks". In *Proceedings of the 44th Annual Southeast Regional Conference*, pages 383–388. ACM, 2006.

[66] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The graph neural network model". *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[67] Franco Scarselli, Sweah Liang Yong, Marco Gori, Markus Hagenbuchner, Ah Chung Tsoi, and Marco Maggini. "Graph neural networks for ranking web pages". *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 666–672, 2005.

[68] Robert E Schapire. "The strength of weak learnability". *Machine Learning, Springer*, 5:197–227, 1990.

[69] Andrea Sgambelluri, Alessio Giorgetti, Filippo Cugini, Francesco Paolucci, and Piero Castoldi. "Openflow-based segment protection in ethernet networks". *Journal of Optical Communications and Networking, Optical Society of America*, 5(9):1066–1075, 2013.

[70] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. "Graph neural networks in particle physics". *Machine Learning: Science and Technology, IOP Publishing*, 2(2):021001, 2020.

[71] Zhaogang Shu, Jiafu Wan, Jiaxiang Lin, Shiyong Wang, Di Li, Seungmin Rho, and Changcai Yang. "Traffic engineering in software-defined networking: Measurement and management". *IEEE access*, 4:3246–3256, 2016.

[72] Fengxiao Tang, Zubair Md Fadlullah, Bomin Mao, and Nei Kato. "An intelligent traffic load prediction-based adaptive channel assignment algorithm in SDN-IoT: A deep learning approach". *IEEE Internet of Things Journal*, 5(6):5141–5154, 2018.

[73] Zhihong Tian, Wei Shi, Zhiyuan Tan, Jing Qiu, Yanbin Sun, Feng Jiang, and Yan Liu. "Deep learning and dempster-shafer theory based insider threat detection". *Mobile Networks and Applications, Springer*, pages 1–10, 2020.

[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". *Advances in neural information processing systems*, 30, 2017.

[75] Shishuang Wang, Hongli Xu, Liusheng Huang, Xuwei Yang, and Jianchun Liu. "Fast recovery for single link failure with segment routing in SDNs". *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 2013–2018, 2019.

[76] Wenjia Wang. "Some fundamental issues in ensemble methods". *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 2243–2250, 2008.

[77] Yongxin Wang, Kris Kitani, and Xinshuo Weng. "Joint object detection and multi-object tracking with graph neural networks". *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13708–13715, 2021.

[78] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. "Transformers in time series: A survey". *arXiv preprint arXiv:2202.07125*, 2022.

[79] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou,

Michael Steinbach, David J. Hand, and Dan Steinberg. "Top 10 algorithms in data mining". *Knowledge and Information Systems, Springer*, 14(1):1–37, 2008.

[80] Junbi Xiao, Song Chen, and Mengmeng Sui. "The strategy of path determination and traffic scheduling in private campus networks based on SDN". *Peer-to-Peer Networking and Applications, Springer*, 12(2):430–439, 2019.

[81] A. Yarkın Yıldız, Emirhan Koç, and Aykut Koç. "Multivariate time series imputation with transformers". *IEEE Signal Processing Letters*, 29:2517–2521, 2022.

[82] Yuan Yuan and Lei Lin. "Self-supervised pretraining of transformers for satellite image time series classification". *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:474–487, 2020.

[83] Murat Yuksel, K. K. Ramakrishnan, and Robert D. Doverspike. "Cross-layer failure restoration techniques for a robust IPTV service". In *2008 16th IEEE Workshop on Local and Metropolitan Area Networks*, pages 49–54. IEEE, 2008.

[84] Chuangchuang Zhang, Xingwei Wang, Fuliang Li, Qiang He, and Min Huang. "Deep learning–based network application classification for SDN". *Transactions on Emerging Telecommunications Technologies, Wiley Online Library*, 29(5):e3302, 2018.

[85] Zezheng Zhao, Chunqiu Xia, Lian Chi, Xiaomin Chang, Wei Li, Ting Yang, and Albert Y Zomaya. "Short-term load forecasting based on the transformer model". *Information, Multidisciplinary Digital Publishing Institute*, 12(12):516, 2021.

[86] Hao Zhu, Yankai Lin, Zhiyuan Liu, Jie Fu, Tat-Seng Chua, and Maosong Sun. "Graph neural networks with generated parameters for relation extraction". *arXiv preprint arXiv:1902.00756*, 2019.

[87] Yuan Zuo, Yulei Wu, Geyong Min, and Laizhong Cui. "Learning-based network path planning for traffic engineering". *Future Generation Computer Systems, Elsevier*, 92:59–67, 2019.

[88] Yuan Zuo, Yulei Wu, Geyong Min, Chengqiang Huang, and Ke Pei. "An intelligent anomaly detection scheme for micro-services architectures with temporal and spatial data analysis". *IEEE Transactions on Cognitive Communications and Networking*, 6(2):548–561, 2020.