DEVELOPMENT OF A NETWORK CONGESTION MANAGEMENT SYSTEM

BY

NWANKWO KENECHUKWU CHRISTIAN

REG. NO: GOU/U22/CSC/809

Project Report Submitted to the Department of Computer Science and Mathematics,
Faculty of Computing and Information Technology,
Godfrey Okoye University, Enugu State

In Partial Fulfillment of the Requirements for the Awarding of the Degree of Bachelor of Science In Computer Science

SUPERVISOR:
Dr. Frank Okebanama

JUNE, 2026

APPROVAL PAGE

The above study, entitled "Network Congestion Management System," has been evaluated and deemed fit for presentation by the Committees of the Departments of Computer Science and Mathematics of the Faculty of Computing and Information Technology, Godfrey Okoye University, Enugu, Nigeria.

	Signature	**Date**
Dr. Frank Okebanama	.....…	
	.....	
Supervisor		
External Examiner	.....…	
	.....	
External Examiner		
Dr. Frank Okebanama	.....…	
	.....	
Sub-dean/HOD, Department of Computer Science and Mathematics		
Prof. I. A. Ajah	.....…	
	.....	
Dean, Faculty of Computing and Information Technology		

CERTIFICATION

In accordance with the University's policies, I hereby declare that all the research conducted as per this paper was done by me, based mainly on my observations and investigations. Therefore, I will take full responsibility for all decisions made by the University regarding this matter..

_____

NWANKWO KENECHUKWU CHRISTIAN
REG. NO: GOU/U22/CSC/809

DEDICATION

This dissertation is dedicated to my parents, Mr & Mrs Ugonna Nwankwo, who through their selfless sacrifices, prayers, and support have enabled me to pursue my studies, as well as all the Nigerians involved in network administration, who strive to keep our network free from congestion.

ACKNOWLEDGEMENTS

In the first place, all praises be to God Most High who graced me and endowed me with wisdom through all stages of this piece of work.

My gratitude will forever remain due to Dr. Uchenna Franklin Okebanama, whose invaluable guidance and constructive criticisms played significant roles in the execution of this study.

To the lecturers and administrative staff members of the Computer Science Department, I am grateful for their contribution towards creating an academic atmosphere which enabled me carry out this work.

To my family members who have consistently stood by me throughout my academic career with words of encouragement, prayers, and sacrifices – my appreciation remains due to you.

Lastly, to my friends and colleagues who contributed ideas and spent countless hours with me during this process, this success belongs as much to you as it does to me.

ABSTRACT

Problems related to the occurrence of network congestion remain among the most important in the area of communication technologies infrastructure because they result in poor quality of services, increased delays and losses and decrease traffic capacity. The inability of reactive solutions of network congestion handling to solve effectively the occurring problems is evident when discussing dynamic networks. This research is devoted to proving how to develop and use a predictive system for detecting the state of network congestion based on XGBoost algorithm. The proposed approach relies on stack architecture where back-end is implemented using Python and Flask and the front-end is designed using React. To conduct the experiment the synthetic data set reflecting performance metrics for networks of different topologies has been used. Such indicators as utilization of bandwidth, packet loss ratio, latency, jitter, queue length and throughput were used to describe network performance. XGBoost classifier model has been built and used to design RESTful API that operates through Flask back-end. The React-based front end includes a simple dashboard that facilitates file upload of CSV format, prediction generation, visual analysis of the prediction results using donut chart representation, and report generation. In implementing this system, a Waterfall strategy was considered, while the system structure and behaviors were captured through UML diagramming techniques, using OOAD principles. According to the experimental results, the GBDT model provides predictions with high accuracy, hence outperforming the rest of the models. The system is, therefore, a viable machine learning tool in addressing the problem of network congestion.

TABLE OF CONTENT

TABLE OF FIGURE

LIST OF TABLE

CHAPTER ONE
INTRODUCTION

1.1 Background of the Study

With the growing use of computer networks and increasing dependence on infrastructures connected to the Internet, there emerge several challenges in network management, monitoring performance, and traffic control. Current enterprise networks are dynamic systems where huge amounts of data packets pass each second via thousands of end devices interconnected through a system of switches, routers, and access points. Network congestion, losses of packets, delays in transferring the data packets, and ineffective routing can result in an inability of organizations to perform their activities effectively, which will cause considerable economic losses.

Currently used methods of network management operate mainly based on predetermined rules and thresholds of the occurrence of some parameters and require human intervention by means of network administrators. Although these methods have been proved to be efficient in smaller networks or less dynamically functioning environments, they become insufficient in current rapidly changing, highly heterogeneous networks. Traffic conditions in the network change constantly due to unpredictable behavior of users and possible cyber attacks from outside the organization. Static configurations cannot react timely to these changes leading to delayed recognition of the congestion and poor routing decisions.

The development of Artificial Intelligence (AI) and Machine Learning (ML) provides innovative opportunities for intelligent network management. The ML models, especially those using ensemble learning like Gradient Boosted Decision Tree (GBDT) and XGBoost, have shown their impressive skills in capturing patterns in the large amount of information in network traffic data, and predicting traffic conditions even before any congestion occurs. By training themselves in a dataset of network traffic data with various metrics including latency, jitter, throughput, buffer occupation, bandwidth occupation, and packet loss rate, the algorithms can classify the traffic and make smart automatic routing decisions for the network.

The system is an AI-based network monitoring and intelligent packet routing solution developed to compensate for the drawbacks of traditional approaches to network

management. This solution uses an ML model built using XGBoost on top of a hybrid dataset consisting of one million records of network traffic data. Other components of The system include a real-time WebSocket data streamer, a React front-end dashboard and a Flask back-end REST API, which together monitor five network switches in real-time, predict the level of congestion risk and initiate smart rerouting including load balancing, path rerouting, and emergency rerouting.

1.2 Statement of the Problem

Modern enterprise-level network infrastructures are under ever-growing pressures due to increasing volumes of traffic, diversification of applications, and zero tolerance for downtime. While networking equipment is advancing by leaps and bounds, network intelligence controlling traffic routing and congestion handling remains reactive and based on rule-driven logic. In practice, it means that network administrators typically become aware of congestion problems only when it's too late, with traffic already degraded and with noticeable packet loss and latency increases experienced by end-users.

This study is prompted by the following specific problems:

1. Currently available monitoring utilities such as SolarWinds, Nagios, and PRTG report data and issues alerts, but they lack intelligent prediction capabilities that could predict traffic congestion in advance.

2. Routing algorithms such as OSPF and RIP cannot be used for dynamically adjusting routing decisions based on actual and predicted traffic situations; they depend entirely on pre-calculated metrics.

3. Network engineers still require a certain amount of time to analyze problems and make decisions, which results in a lag between congestion occurrence and resolution.

4. AI-driven network management systems are either overly complex for implementation on regular hardware infrastructure or fail to incorporate traffic monitoring and routing into one solution.

5. There is a need for an open and deployable system that uses machine learning to predict congestion and visualize and route automatically at real-time speeds.

6. All these challenges have necessitated the development of a system called NetSentinel that connects the dots between network monitoring and intelligent traffic management.

1.3 Aim and Objectives of the Study

The aim of the research is to develop a network congestion management system that utilizes machine learning techniques to forecast network congestion in real time and perform optimized routing action. For the above-mentioned goal to be achieved, the following specific objectives should be met that is to:

1. Evaluate current network monitoring and routing systems' limitations and define the needs for an alternative solution based on AI technologies.

2. Design and develop an XGBoost machine learning model based on a hybrid network traffic dataset of one million records to forecast congestion and classify routing decisions in real time.

3. Implement a Flask-based RESTful API backend to link the ML model with a live WebSocket-powered network traffic generator.

4. Design and implement a responsive React frontend dashboard to visualize network topology, monitor switches, manage packet logs and generate analytics reports.

5. Test the performance of the system in terms of machine learning accuracy and system efficiency in congestion forecasting and routing actions implementation.

1.4 Significance of the Study

The relevance of this research lies in multiple areas ranging from academics, technical implementation, and practical applications. On an academic level, this research makes contributions to the emerging area of AI-powered networking, presenting a case for utilizing

ensemble machine learning in practice for creating a deployable full-stack networking solution.

From a technical point of view, the system presents a unique combination of such features as XGBoost-based prediction, augmentation of decision rules, real-time data stream via WebSockets, and interactive multi-page dashboard in one single product. Its architecture can serve as a guide for implementing similar systems at other institutions in need of modernizing their networking center operations.

On a practical level, the system is particularly relevant for universities, governments, SMEs, telecommunication companies, etc., which operate local area networks and seek an efficient and affordable monitoring and routing solution. By enabling the process of automated congestion management and routing, the system helps reduce burden on IT departments, ensures minimal network interruptions, and helps adopt proactive measures instead of reacting to potential threats. The system also generates useful information such as risk distribution statistics, most congested switch identification, and historical packet logs review for making decisions.

In addition, this research shows that undergraduates can develop functional software systems to solve technology problems in the real world and set standards for future computer science researches at Godfrey Okoye University and other educational institutions alike.

1.5 Scope of the Study

Functional boundaries of this research include the creation and development of a network monitoring solution that includes congestion prediction in real-time, risk management, routing decision process automation, and multi-view analytics dashboard. The solution monitors five virtual network switches SW1 to SW5 and implements packet routing between virtual end nodes (PC1 to PC25).

Technical boundaries of this research consist of implementing software solutions for backend, API, real-time stream, machine learning models, and database using Python 3, Flask and Flask-SocketIO, XGBoost, and SQLite respectively. Frontend implementation is done using

React with Vite framework, while communication between the server and clients is performed by Axios and Socket.IO.

The machine learning model uses 12 numerical features including latency, jitter, throughput, queue length, buffer occupancy, bandwidth utilization, packet loss rate, hop count, cost of primary and alternative routes, load at the switch, and failover flag, and two categorical features such as primary switch and secondary switch.

This research does not consider network deployment, deep packet inspection, and integration with external network orchestration platforms like Cisco DNA Center and OpenDaylight. Network simulation in this project is driven by datasets and does not include any live packet capturing through physical network interfaces. It is developed and tested on a Windows development platform and is expected to run on conventional server computers without using any GPU capabilities.

1.6 Limitation of the Study

Nevertheless, there are still a number of limitations to this study that need to be pointed out:

1) Dataset Limitations: The neural network is built only on the basis of the hybrid network dataset of one million records. Although it allows us to cover a wide array of network cases, the individual features of the traffic pattern of some network environment might not be covered by the dataset. In the case of implementation in a live network, there will be a need to re-train the neural network with the particular environment dataset.
2) Simulator-Driven Implementation: The implemented system is not tested on the actual traffic but through a simulator that works based on the dataset provided for simulation. As a result, no novel traffic situations can be tested due to the simulator's nature.
3) Single Machine Implementation: The whole system is run on the basis of a single machine and, thus, cannot provide an understanding of how the architecture works in a multi-machine network setting.
4) Security Hardcoded: At present, this system employs security that is completely hardcoded using secret keys and an entirely unrestricted CORS policy that will not work

in a production environment until changes are made. Improving the security level of the system was out of scope for this undergraduate project.

5) Not Implemented for Physical Networks: This system is not designed to communicate with any physical networking devices and cannot perform live captures via Wireshark or tcpdump.

1.7 Operational Definition of Terms

Definitions of the following terms are provided based on their meaning within the context of the study:

Artificial Intelligence (AI): The replication of human brain functions like learning, reasoning, and problem solving using computers, in this case applied to network traffic analysis and routing decisions.

Machine Learning (ML): The area of artificial intelligence where computers learn by recognizing patterns and make decisions or predictions without being explicitly programmed to do so. For instance, in the current study, XGBoost is used to classify congestion and predict routing decisions.

XGBoost (Extreme Gradient Boosting): An efficient and highly scalable machine learning library based on gradient boosting methods for fast and accurate predictions. This library forms the main basis of the current research in predicting congestion and routing decisions.

Network Congestion: A situation when a node or link on a network is being bombarded with more information than it can handle. This results in queuing of packets, latency, jitter, and eventually, packet loss.

Bandwidth Utilization: The measure of how much the total bandwidth on a particular network is being utilized, measured in terms of percent between 0 and 100% at each switch level.

Packet Rerouting: The action of routing network traffic to an alternate route to overcome congestion or bottlenecks in the existing route. In this particular case, the routing decision is made automatically via the ML engine depending on the risk level and predicted congestion.

Risk Levels: A classification of network risk status into four classes based on utilization levels such as Low (0-40%), Medium (41-70%), High (71-85%), and Critical (86-100%) utilized to decide routing strategy and urgency.

WebSocket: A two-way protocol for establishing full-duplex communication channels on top of a TCP connection that will be used to feed real-time network data from the backend simulation system to the frontend display.

Flask: A light Python library that is used to implement the backend part of the application that utilizes the RESTful architecture.

React: A library of components for constructing user interfaces written using JavaScript, used in this project for creating the interface of the system

SQLite: A light database used for storing rerouted packets information and user authentication data.

REST API (Representational State Transfer Application Programming Interface): A software design pattern used for developing network-based applications. For this application, REST API provides an endpoint that offers features of authentication, predictions, logging, and analysis.

Traffic (Network Traffic): The total volume of data being sent via the network during a certain period of time, in Mbps.

Latency: The time gap in terms of transmission of data through the network, in milliseconds (ms).

Queue Length: The total count of data packets stored in the buffer prior to the transmission through the network, signifying the extent of the load on the network.

Bandwidth: The total capacity of the network in sending data, in Mbps.

Packet Loss: The proportion of data packets that do not reach their destination point.

Jitter: The gap in data packet arrival time.

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

The current chapter aims at carrying out a thorough evaluation of the existing literature that is pertinent to the research topic of using Artificial Intelligence for effective network monitoring and intelligent packet routing systems. The importance of the present literature review lies in assisting the researcher and the reader understand the history, developments, and drawbacks associated with other studies in the field of AI and machine learning-based network management and congestion prediction systems. In addition to this, the review will serve to provide justification for the current research effort, which can be achieved through analyzing the current achievements in the field.

This part includes a critique of scientific journals, research papers, and conference proceedings to point out the advantages and weaknesses of other researchers' works and to uncover knowledge gaps covered by the proposed system. This literature review chapter consists of four sections, namely the theoretical background of the research, a review of other scholarly works, comparison of the research findings in tabular form, and a research gap section.

2.2 Theoretical Background

The following discussion will revolve around the basic theories, principles, and concepts that form the basis of the system. This sets up the background against which the approaches and models used in the research can be understood.

2.2.1 Overview of Computer Network Traffic and Congestion

Network traffic is defined as the amount of data flowing through a network at any point in time. Some of the key indicators of traffic include throughput (data transfer success rates), latency (latency time), jitter (variations in latency times), queue length (number of data packets in line for processing), buffer usage (percentage of buffers utilized), and loss rate (number of data packets lost). Network congestion refers to situations where there is an

overload of data on one or multiple nodes in the network system. This often results in longer queue lengths, high latency, excess jitter, and eventually packet losses. Causes of network congestion include unexpected surges in network traffic, DDoS attacks, breakdowns, and constant overload of certain links. Economic implications of network congestion without control include poor QoS (quality of service), SLA violations, and reduced revenues, especially in cases of latency sensitive applications like VoIP, video conferencing, and financial transactions.

2.2.2 Traditional Network Monitoring and Routing Methods

The most commonly used protocol-based solutions for network monitoring include SNMP, NetFlow, and ICMP. These protocols gather data about network metrics, which are analyzed using third-party software, like Nagios, PRTG Network Monitor, Zabbix, or SolarWinds Network Performance Monitor. Even though these solutions allow one to see how a network is working, they only detect issues after a specific threshold was exceeded. Thus, congestion or failure has already happened at that point when alerts are generated.

In terms of routing technologies, traditional dynamic routing protocols like OSPF and EIGRP calculate paths based on administrative costs of network components, such as link bandwidth or hop count. They adjust their operation when the network topology changes but lack the ability to account for traffic loads and congestion prediction capabilities. Traffic engineering features in MPLS allow a more advanced approach to path calculation, but configuring such systems requires special equipment and knowledge.

2.2.3 Machine Learning and Its Applications in Network Management

Machine Learning is one of the branches of artificial intelligence that allows machine learning models to be capable of identifying patterns in data and improving themselves without explicit programming. Some of the applications of ML to problems related to network management include anomaly detection, intrusion detection, traffic classification, Quality of Service prediction, and congestion management.

For instance, supervised learning algorithms are suitable for congestion prediction and classification of routing decisions. Of all types of supervised learning algorithms, ensemble-

based tree algorithms are especially effective in classifying structured data such as network traffic. Random Forest, GBDT, and XGBoost provide both high prediction accuracy and interpretability compared to deep neural networks, which makes them preferable for operational network management solutions where interpretability is considered valuable.

2.2.4 XGBoost and Gradient Boosted Decision Trees

XGBoost is a machine learning algorithm proposed by Chen & Guestrin (2016) that implements a highly scalable and computationally efficient version of gradient boosting and offers state-of-the-art performance across various benchmark problems. The model creates an ensemble of decision trees that are sequentially added one after another, with each new tree compensating for the weaknesses of the previous models through learning residual gradients. Notable features of XGBoost include support for missing values, regularization against overfitting, and parallel tree building.

When it comes to network traffic classification, XGBoost has been consistently shown to perform better than conventional classification algorithms like Naive Bayes or SVM and even compete effectively with modern neural networks when the data format is tabular as opposed to images or audio. In addition to high dimensionality, network traffic data is known to include both numerical and categorical variables, making it suitable for processing by XGBoost on the twelve-feature network traffic dataset.

2.2.5 Software-Defined Networking and Intelligent Routing

Software Defined Networking (SDN) is a design methodology that allows separation of the network control plane and data plane, thereby facilitating centrally controlled and programmatically driven network management. SDN controllers like OpenDaylight and ONOS are capable of installing flow rules on network devices based on policy-based decisions, thus offering an ideal platform for machine learning-based routing. Several studies have proven that the incorporation of machine learning capabilities into SDN controllers results in dynamic and responsive traffic engineering capabilities that outperform traditional routing algorithms.

While the physical SDN architecture is not available at present, the same concepts related to centrally driven routing policies using machine learning can be simulated using software, as demonstrated in this project via the system backend routing algorithm that makes routing decisions and logs them as if they were going to be translated into actual SDN flow rules.

2.2.6 Real-Time Data Streaming and WebSocket Communication

Real-time network monitoring needs communication frameworks that can provide telemetry data updates to users' interfaces at a very high frequency and with lower latency. HTTP requests are not suitable for this task because they cause more overhead and generate more traffic than required; moreover, the delay in HTTP polling requests is directly related to the frequency of requests. WebSocket framework, specified in RFC 6455, ensures a persistent, two-way communication between the client and the server via a single TCP connection. Flask-SocketIO is the WebSocket framework used in this project, which supports the implementation of event-driven and real-time communication between the client and the server through Socket.IO.

2.3 Review of Related Works

This part will analyze the existing research on the subject of artificial intelligence-based network monitoring, traffic congestion forecasting, and routing strategies. The analysis covers seventeen pieces of relevant literature published between 2008 and 2022.

Nguyen and Armitage (2008) undertook a pioneering survey on machine learning methods in network traffic classification, discussing over fifty papers. The authors found out that the most accurate results could be obtained through supervised learning algorithms with decision tree-based classifiers being most efficient. In addition, it was highlighted in the survey that feature selection is crucial for the performance of the classifier used, which explains why the system applies fourteen highly predictive features, the most important of which have been recognized to be bandwidth_utilization, switch_load, and packet_loss_rate.

Benson et al., (2010) performed an empirical study of network traffic attributes in data center networks by examining network traffic trends in ten different data centers. They found out that bandwidth utilization was the sole predictor of network congestion, being the cause for

most packet loss and latency degradation events. This empirical observation serves as an academic justification for the risk assessment logic used in the system's function, where bandwidth utilization is used as the sole criterion to determine the risk tier.

Pensieve from Mao et al. (2017) was a neural network based adaptive bit rate video streaming scheme that relied on reinforcement learning for routing decision-making. While Pensieve delivered impressive performance in optimization of path selection within the network, the scheme was limited to video streaming, and required substantial training time before being able to learn through reinforcement learning. On the other hand, the NetSentinel application relies on supervised learning that allows for quick inference and does not require the simulation of the environment in the training process.

Neural Packet Classification, designed by Suresh et al. (2017), relied on fully connected neural networks to implement deep learning for classification of network traffic. The approach proved to be highly accurate when employed for the CAIDA data set, although high computation requirements necessitated the use of GPU acceleration for inference. In comparison, the system delivers the same level of accuracy using the XGBoost classifier on CPUs.

Traffic Engineering Architecture proposed by Xie et al., 2018 made use of a prediction model using GBDT to determine link utilization for proactive rerouting as well as minimization of network latency. Their system utilized OpenDaylight SDN Controller and showed improvement of 15% in terms of mean network delay. Nevertheless, this work only focused on physical implementation of SDN architecture without incorporating a user-friendly monitoring system. With this system both aspects have been taken care of by offering a comprehensive solution integrating machine learning inference, routing and visualizations.

Azzouni and Pujolle, 2017 introduced NeuRoute, a recurrent neural network based approach aiming at traffic matrix prediction for proactive routing of data packets. Despite the efficiency offered by LSTM network employed in traffic prediction, high computational complexity hinders the practical applicability of this method in reality. Such a challenge has been overcome in the system through employing XGBoost algorithm for inference purposes,

which provides the same intelligent routing capabilities while keeping computational cost minimal.

Fadlullah et al. (2017) introduced an intelligent traffic control system based on a deep neural network, illustrating that ML-enabled traffic control decreases average packet delays up to 30% better than traditional routing algorithms. Although the system performed extremely well, it needed hardware acceleration through GPUs in order to train and perform inference as well as lacked interpretability features from the perspective of network operators. This research confirms the feasibility of using machine learning for intelligent traffic control purposes, although it was constrained by hardware dependencies and interpretability, which influenced the choice of XGBoost in the system's design.

Finally, Boutaba et al. (2018) conducted a review about the applications of machine learning techniques in network management. In particular, the paper discussed the use cases of machine learning for traffic classification, routing optimization, resource management, fault management, and quality of service predictions within the scope of both conventional and software-defined networks. Specifically, the authors revealed that supervised learning techniques were able to achieve significantly better results than rule-based algorithms when it came to analyzing network telemetry data, as well as a lack of deployable ML-based inference systems in place.

The research by Zhang, Patras, and Haddadi (2018) focused on investigating deep learning usage in mobile and wireless networks regarding applicability of CNN and LSTM models in traffic classification and QoS prediction problems. The results obtained by researchers indicated the possibility of achieving accurate classification with the use of such machine learning methods but revealed the necessity of using large computational capacity and significant amount of training data, which made such models not feasible for deployment on regular network devices. Therefore, these findings support the choice of architecture of the system where XGBoost is utilized rather than deep learning.

Amarasinghe, Kenney, and Manic (2018) introduced a framework to make deep neural network anomaly detection systems more understandable. They showed that the interpretability of ML solutions is critical since it increases operators' confidence in model decisions and reduces the time spent on approving its suggestions. The key point here is that

no matter how precise black-box models are, they are always turned down for being unable to provide a justification for the decision. In contrast, the system meets this criterion by presenting a wide range of interpretability information on the Analytics page.

In their research article titled "Random Forest-based QoS-aware Routing for Voice Video Communication", Liu et al. (2019) introduced an innovative QoS-aware routing framework that leveraged the Random Forest classification algorithm to select appropriate network links for better quality service. This technique proved effective in improving quality service in voice and video communications; however, it had several limitations since it relied on trace-driven simulations without any real-time stream capabilities or visualisation module.

Similarly, Stampa et al. (2017) applied deep reinforcement learning algorithms in routing optimisation by demonstrating that reinforcement learning algorithms are capable of adapting the routing policies based on the traffic pattern. Nonetheless, this technique involved lengthy convergence times and required the existence of a live computer network for training purposes. In comparison, the system avoids both problems by using offline training based on supervised learning of data stored in a one million records pre-labeled dataset.

The authors Shi et al. (2021) discussed their traffic management model based on XGBoost's application in identifying congestion in vehicular networks. They provided a rationale for using XGBoost through empirical testing that showed the algorithm's efficiency in classifying traffic states with more than 95% accuracy. However, the findings could be applied to vehicular networks only, and did not include any monitoring dashboards, machine learning model explainability visualizations, or automatic routing decisions.

The paper Kato et al. (2017) investigated the use of deep learning for communication problems. In their research, the authors reviewed various applications of neural networks like CNN, RNN, and autoencoder to classify traffic, detect faults, and establish optimal routing paths. Importantly, they emphasized the inefficiency of deep learning techniques since they were not computationally efficient nor interpretable, which influenced the system's choice of XGBoost algorithm.

The intelligent routing system presented by Xiao et al. (2019) employs an ensemble classifier consisting of Random Forest and Gradient Boosting classifiers for Software Defined

Networks, yielding superior path selection accuracy through voting among several classifiers. This research shows that ensemble methods yield better results than non-ensemble models when applied to routing classification problems. This provides strong support for using ensembles of decision trees like the one implemented in NetSentinel XGBoost algorithm, as well as the superiority of gradient boosting techniques for dealing with network telemetry data.

Hui et al. (2021) proposed an autonomous deep reinforcement learning framework for managing 5G heterogeneous networks. The authors showed that the use of RL agents allowed adjusting resource allocation and routing decisions according to the dynamics of traffic demand changes. Although RL yielded excellent results in 5G settings, it was necessary to learn from real interactions with the environment continuously. This implies that RL cannot be trained off-line and cannot be used in the system, as offline learning algorithms are needed to deploy the algorithm from a pre-trained model immediately without interacting with the network.

In a paper by Jain et al. (2022), a study was carried out to evaluate the effectiveness of explainability techniques in machine learning models applied to intrusion detection systems. The comparison was between the performance of SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-Agnostic Explanations) with the native feature importance technique for XGBoost models. The results indicated that the feature importance native to XGBoost is both effective and computationally efficient in explaining the structured data of network telemetry and is therefore the best XAI method for the operational environment of the network management system.

2.4 Summary of Literature Review

Table 2.1 below presents a comparative summary of all seventeen related works reviewed in this chapter, highlighting their contributions, techniques, limitations, and specific relevance to this system.

S/N	Author(s)	Contribution	Technique	Limitation	Relevance to NetSentinel
1	Nguyen & Armitage (2008)	Landmark survey of ML for network traffic classification — reviewed 50+ studies; established decision trees as most accurate for structured telemetry data.	Survey (Decision Trees, Naive Bayes, SVM)	Survey only; no deployable system produced.	Provides foundational academic grounding for supervised learning approach; validates feature selection importance applied in NetSentinel.
2	Benson et al. (2010)	Empirical study of traffic in 10 data centres confirming bandwidth utilization as the primary	Empirical measurement and analysis	Measurement study only; no ML model or automated routing.	Validates bandwidth_utilization as primary input in calculate_risk_level; confirms NetSentinel's risk tier thresholds.

		indicator of congestion.			
3	Mao et al. (2017)	Pensieve — RL-based adaptive bit-rate CDN path selection demonstrating ML-driven routing optimisation.	Deep Reinforcement Learning	Narrow scope (video only); long RL training time.	Validates ML routing concept; highlights advantage of supervised XGBoost for faster inference.
4	Suresh et al. (2017)	Neural Packet Classification with fully connected NNs on CAIDA dataset for multi-class traffic type identification.	Deep Neural Networks	Requires GPU; high computational cost.	Confirms deep learning accuracy; justifies XGBoost for CPU-deployable production systems.
5	Xie et al. (2018)	SDN traffic engineering	GBDT + SDN (OpenDaylight)	Requires physical SDN	Closest predecessor; validates GBDT for

		with GBDT for link utilisation prediction; 15% delay reduction achieved via OpenDaylight.		hardware; no monitoring dashboard.	routing. NetSentinel adds full-stack dashboard and explainability.
6	Azzouni & Pujolle (2017)	NeuRoute: LSTM-based proactive routing using traffic matrix prediction in SDN environments.	LSTM / Recurrent Neural Network	High complexity; significant training overhead.	Demonstrates proactive routing value; NetSentinel achieves similar goals with lighter XGBoost model.
7	Fadlullah et al. (2017)	DNN-based intelligent traffic control for heterogeneous networks; achieved up to 30% packet	Deep Neural Networks	Requires GPU; no model interpretability for operators.	Validates ML-driven traffic control; confirms hardware and interpretability limitations that favour XGBoost in NetSentinel.

		delay reduction.			
8	Stampa et al. (2017)	Deep RL for dynamic routing policy optimisation in computer networks demonstrating adaptive routing agents.	Deep Reinforcement Learning	Long convergence time; requires live network environment for training.	Validates adaptive routing concept; offline supervised training in NetSentinel avoids convergence constraints.
9	Boutaba et al. (2018)	Comprehensive survey of ML in network management covering traffic classification, routing, QoS, and fault management. Identified lack of integrated deployable	Survey (Supervised and Unsupervised ML)	Survey only; no deployable system produced.	Directly identifies the gap that NetSentinel fills — an integrated, deployable, ML-powered network management platform.

		systems as a critical gap.			
10	Zhang et al. (2018)	Comprehensive study of deep learning in mobile and wireless networking covering CNN and LSTM for traffic classification and QoS prediction.	CNN, LSTM (Deep Learning)	Requires GPU; high computational cost for real-time deployment.	Reinforces XGBoost selection over deep learning for CPU-deployable production systems in NetSentinel.
11	Amarasinghe et al. (2018)	Explainability framework for DNN-based anomaly detection in industrial networks showing that	XAI — SHAP, LIME, Feature Importance	Industrial domain only; not generalised to enterprise LAN network management.	Validates explainability requirement addressed in NetSentinel's Analytics page through feature importance,

		interpretable ML increases operator trust and reduces response time.			confusion matrix, and ROC curve.
12	Liu et al. (2019)	ML-based QoS routing for multi-service networks. Random Forest predicted link quality for path selection for voice and video.	Random Forest Classifier	Offline evaluation only; no real-time streaming or dashboard.	Supports ensemble ML for routing; NetSentinel extends with real-time WebSocket streaming and live visualisation.
13	Xiao et al. (2019)	Ensemble routing system using combined Random Forest and Gradient Boosting classifiers for	Random Forest + Gradient Boosting Ensemble	No real-time deployment; no monitoring dashboard.	Validates ensemble gradient boosting for network routing, directly supporting XGBoost selection in NetSentinel.

		SDN path selection; showed ensemble outperforms single models.			
14	Shi et al. (2021)	XGBoost congestion detection in vehicular networks achieving >95% accuracy from network telemetry data.	XGBoost Classifier	Vehicular domain only; no dashboard, explainability, or routing module.	Directly validates XGBoost for congestion classification; NetSentinel generalises to enterprise LAN with routing and explainability.
15	Hui et al. (2021)	Deep RL-based autonomous network management framework for 5G heterogeneous	Deep Reinforcement Learning	Requires continuous online learning from live network; not deployable from pre-	Confirms adaptive network management value; NetSentinel's offline training enables immediate deployment.

		networks demonstrating dynamic resource allocation.		trained model.	
16	Kato et al. (2017)	Comprehensive survey of deep learning for networking covering CNNs, RNNs, and autoencoders for traffic classification and routing optimisation.	Survey (CNN, RNN, Autoencoders)	Deep learning interpretability and hardware requirements limit operational deployment.	Validates ML-in-networking trend; reinforces ensemble methods over deep learning in production systems.
17	Jain et al. (2022)	Comparative study of XAI methods (SHAP, LIME, feature importance) for	XGBoost Feature Importance (Gain), SHAP, LIME	Intrusion detection domain; not generalised to congestion prediction or	Validates gain-based XGBoost feature importance implemented on NetSentinel Analytics page as the

		ML in network intrusion detection confirming native XGBoost gain-based importance is most practical for structured telemetry.		routing.	appropriate XAI method.

Table 2.1: Summary of Related Works in AI-Powered Network Management and Routing (17 Works)

2.5 Research Gap

However, the literature review shows that despite many works having been done regarding ML techniques for network traffic classification, congestion prediction, and routing optimization, there still exist some gaps which will be filled with this project.

Firstly, most of the works have studied the ML modeling part separately or the routing part separately but none of them has created a complete software package for network optimization that can be deployed and used immediately. Among all, the work that was closest to this project was the work of Xie et al. (2018), which required SDN hardware and did not have any dashboard for monitoring. The system provides a complete solution including the part of ML model and live network visualization.

Secondly, real-time visualization of the current status of the network including congestion, network topology and analytical dashboard has not been provided in most of the academic papers, although tools like Nagios and PRTG have already been developed for real-time network management. This system fills this gap by integrating live WebSocket React dashboard into the network.

Third, previous works mainly adopt a deep learning approach (e.g., NNs, LSTMs, and CNNs) which, despite being highly effective, poses a considerable challenge when it comes to infrastructure needs. Previous research shows (Shi et al., 2021; Xie et al., 2018) that gradient boosting algorithms, such as XGBoost, are capable of achieving high predictive accuracy on the network telemetry data with considerably lower costs. This system capitalizes on this insight to provide a solution that can be deployed on general-purpose computing devices without using GPUs.

Fourth, none of the studies analyzed tackles the problem of implementing two types of model predictions (congestion prediction and routing action prediction) at the same time, which NetSentinel's ML engine accomplishes. This system simultaneously predicts whether there is network congestion and what the routing action (NORMAL, REROUTE, or EMERGENCY_REROUTE) ought to be.

This research aims to fill these gaps by offering an open, fully stackable, machine learning-based system for network monitoring and intelligent routing.

CHAPTER THREE
SYSTEM ANALYSIS AND DESIGN

3.0 Introduction

In this chapter, we outline the research methodology and design for our project, NetSentinel. This includes an outline on research methodology, ML methodology (CRISP-DM), software design methodology (OOAD), systems analysis, requirements analysis, dataset details, data preparation techniques, UML modeling, database design, systems architecture, evaluation approach, and technology used.

3.1 Research Design and Methodology

3.1.1 Research Design

The current research adopts a Quantitative Experimental research methodology. This is because the research yields numerically measurable performance results such as Accuracy, Precision, Recall, F1-Score, and AUC-ROC based on a held-out dataset. Also, this research is quantitative since performance differences between the proposed dual-model NetSentinel XGBoost architecture and the rule-based baseline approach are analyzed using quantitative measures. The data employed consists of one million records of hybrid network traffic that have been equally partitioned into training and testing sets.

3.1.2 Software Design Methodology — OOAD

OOAD was chosen as the software design method for NetSentinel. This is the best choice since the whole back end is developed using Python, which is an object-oriented programming language, where there are well-defined objects bearing responsibility such as NetworkSimulator, MLEngine, Flask Application, and SQLAlchemy Model Classes (User, ReroutedPacket). OOAD has been done by use of UML, giving the following results: Use case diagram, Activity Diagram (Existing/Proposed), System Architecture Diagram, Class Diagram, and Entity-Relationship Diagram.

3.1.3 Justification for Methodology

The OOAD methodology was chosen for designing the system due to the following reasons. To begin with, the system is composed of clear object types such as User, Rerouted Packet, Network Simulator, and other ML engine objects. This makes OOAD suitable in developing a model which reflects the objects in the problem domain since the objects in the problem domain directly translate into object classes in the system.

In addition to that, the focus of OOAD on encapsulation and modularity helps satisfy the architectural constraint that calls for independence of modules such as the ML engine, database layer, API and frontend. As such, each module can be separately designed, tested and modified without affecting other parts of the system.

Last but not least, the UML modeling features provided by OOAD such as use case diagrams, activity diagrams and class diagrams have proven useful in representing system behaviors.

The Cross Industry Standard Process for Data Mining (CRISP-DM) was selected as the basis of the ML process framework for this project. The CRISP-DM process is iterative and repeatable and encompasses all stages of the ML process, starting with problem identification and ending at implementation (Chapman et al., 2000).

3.2 System Analysis

3.2.1 How the Existing System Works

The existing management of enterprise networks makes use of two concurrent techniques. Firstly, the hardware routing protocol, which includes OSPF, EIGRP, and RIP, uses static cost metrics to identify shortest path and install the route in the forwarding table of switches. Secondly, SNMP-based management systems like Nagios, PRTG, Zabbix, and SolarWinds gather device metrics and use rules based on threshold limits for sending an alert signal. Network engineers have to check the alerts manually, determine the causes of congestion, and take corrective actions accordingly. The data regarding the existing system was obtained from secondary sources in Chapter Two.

3.2.2 Limitations of the Existing System

1. Not Proactive But Reactive: The monitoring and reporting tools detect and inform of congestion only once it happens. There are no predictive and preventive capabilities.
2. Reaction Delay: It takes time for humans to respond to events. Packet losses due to congestion happen much more quickly than a person can act.
3. Inflexible Static Routing: OSPF and EIGRP determine routing by computing routes using static administrative weights pre-assigned to interfaces during configuration.
4. Lack of Coordination Between Systems: The monitoring system works separately from the routing algorithm and does not have an intermediate intelligence layer.
5. Lack of Predictive Capability: The tools do not provide any forecasting or risk detection; they merely show current metric values.
6. Lack of Explainability: Currently, no tool shows a confusion matrix, feature importance, ROC curve, or any other indicator of ML algorithms' performance.

3.2.3 Analysis of the Proposed System (NetSentinel)

All these six limitations have been effectively resolved by incorporating NetSentinel's integrated dual model ML approach within a streaming architecture that uses the WebSocket protocol. The XGBoost Congestion Prediction Model learns patterns from one million labeled traffic records, which allows for proactive predictions using machine learning instead of setting static thresholds. The Routing Decision Model automatically decides what should be done (either NORMAL, REROUTE, or EMERGENCY_REROUTE) based on its calculations, thereby removing the need for manually taking routing decisions. Gradations in terms of risk levels – Low, Medium, High, or Critical – are assigned to each case.

3.3 System Requirements Specification

3.3.1 Functional Requirements

FR No.	Requirement	Description
1	User Authentication	Accept username and password; issue JWT tokens; support

		Admin and Analyst roles with access enforcement on all protected endpoints.
2	Real-Time Telemetry Streaming	Stream network telemetry from the NetworkSimulator to all connected frontend clients via WebSocket every two seconds.
3	Feature Preprocessing	Apply StandardScaler to numerical features and OneHotEncoder to categorical features within a scikit-learn Pipeline; persist fitted objects to pickle files.
4	Congestion Prediction	Apply the trained XGBoost binary classifier to incoming feature vectors to predict congestion (0 = none, 1 = congested).
5	Risk Level Determination	Classify network state into Low (0–40%), Medium (41–70%), High (71–85%), or Critical (86–100%) based on bandwidth_utilization.
6	Routing Decision	Combine ML congestion prediction and risk level to determine NORMAL, REROUTE, or EMERGENCY_REROUTE action.
7	Rerouting Event Logging	Persist all non-NORMAL routing events to SQLite with full details: packet_id, nodes, switches, paths, action, reason, and timestamp.
8	Network Topology Visualisation	Display animated live topology with SW1–SW5 and PC1–PC25; colour-coded path lines updated on every WebSocket event.
9	Per-Switch Statistics	Show per-switch utilisation, latency, loss, and jitter in real-time cards with colour-coded status badges (Normal, Warning, High, Critical).
10	Live Trend Charts	Display live-updating charts for Bandwidth/Buffer (%), Latency/Jitter (ms), and Packet Loss (%) over the last 40 readings.

11	Current Snapshot Panel	Display current Bandwidth Usage, Buffer Utilization, Latency, Packet Loss, Jitter, and Queue Length updated every WebSocket event.
12	Live Network Feed Table	Stream individual packet events in a table showing TIME, SRC IP, DST IP, PROTO, PATH, LATENCY, BW%, BUFFER%, LOSS%, JITTER, RISK, DECISION.
13	Packet Log Management	Provide a filterable, searchable table of all rerouting events on the Packet Logs page, filterable by action type.
14	Analytics Reporting	Provide risk distribution donut chart, congestion trend chart, most congested switches bar chart, XGBoost performance chart, and model configuration table.
15	ML Explainability	Display feature importance chart (14 features ranked by gain), confusion matrix (TP/FP/FN/TN), ROC curve (AUC), and Precision-Recall curve (AP).

Table 3.1: Functional Requirements Specification

3.3.2 Non-Functional Requirements

NFR	Category	Requirement
1	Performance	ML inference pipeline shall complete within 200ms of receiving a feature vector. WebSocket interval shall not exceed 2 seconds.
2	Accuracy	XGBoost congestion model shall achieve minimum 95% Accuracy, Precision, Recall, and F1-Score on the held-out test set.
3	Reliability	NetworkSimulator shall run continuously, restarting the dataset loop automatically. Database exceptions shall be handled with rollback.

4	Usability	All six dashboard pages shall be accessible from a persistent navigation sidebar without specialist network engineering knowledge.
5	Maintainability	Clear separation of concerns across app.py, routes.py, ml_engine.py, simulator.py, and models.py. New ML models replaceable via pickle file swap.
6	Security	All endpoints except /api/login shall require a valid JWT Bearer token. Passwords stored as Werkzeug salted hashes only.

Table 3.2: Non-Functional Requirements Specification

3.4 Dataset Description and Selection

Hybrid Network Traffic Dataset

("hybrid_network_dataset_1M.csv") This is a network traffic dataset of one million data entries simulating an enterprise network with five switches under all possible levels of congestion. This network traffic dataset has been chosen since it comes with comprehensive labeling to fit the binary classification of congestion as well as four-class routing decisions; and one million data entries are enough to train XGBoost classifiers.

Statistic	Value
Total records	1,000,000
Network topology	Five switches (SW1–SW5), twenty-five endpoint nodes (PC1–PC25)
Numerical features (12)	latency, jitter, throughput, queue_length, buffer_occupancy, bandwidth_utilization, packet_loss_rate, hop_count, path_cost_primary, path_cost_alternate, switch_load, failover_flag
Categorical features (2)	primary_switch (S1–S5), secondary_switch (S1–S5)

Binary target	congestion_label (0 = no congestion, 1 = congestion)
Multi-class target	routing_decision (NORMAL,REROUTE, EMERGENCY_REROUTE)
Training set	800,000 records — 80% stratified split
Test set	200,000 records — 20% stratified hold-out for evaluation
Simulator feed	live_feed_dataset.csv — exported 20% test split replayed by NetworkSimulator
Missing values	None

Table 3.3: Dataset Summary Statistics

3.5 Dataset Description and Preprocessing

The machine learning algorithms used in NetSentinel have been trained using a hybrid network dataset that comprises one million traffic records (hybrid_network_dataset_1M.csv). The dataset represents traffic data from a five-switch network that generates traffic records under varying levels of congestion and different routing strategies.

Some of the features included in this dataset are: latency (milliseconds), jitter (milliseconds), throughput (megabits per second), queue length (packets), buffer occupancy (as %), bandwidth utilization (as %), packet loss rate (as ratio between 0.0 and 1.0), hop count (as integer), path cost primary, path cost alternate, switch load (as %), failover flag (as binary value: either 0 or 1), primary switch (as categorical: S1-S5), secondary switch (as categorical: S1-S5).

The preprocessing process took place inside the scikit-learn Pipeline module. An 80% / 20% train/test split was done where we used the StratifiedSplit method in order to maintain the balance of our classes (congested vs. non-congested) during the train/test partitioning process.

For the numerical features, normalization was performed using the StandardScaler class (mean 0, standard deviation 1). Categorical features (primary_switch and secondary_switch) underwent the process of encoding through OneHotEncoder, where we set the parameter handle_unknown='ignore' in order to prevent pipeline crashes due to unknown switches at test time.

3.6 System Design — UML Models

3.6.1 Use Case Diagram

There are two players in interaction with NetSentinel: the Authenticated User (Admin) carries out: Login, View Dashboard, View Monitoring, View Topology, Browse Packet Logs, and View Analytics (Risk Distribution, Congestion Trend, Top Five Congested Switches, XGBoost Classification Performance, Feature Importance, Confusion Matrix, ROC Curve, Precision-Recall Curve), while the Network Simulator (system player) carries out: Stream Telemetry Data over WebSocket, Classify Congestion, Routing Decision, and Logging of Rerouting Event to SQLite database.

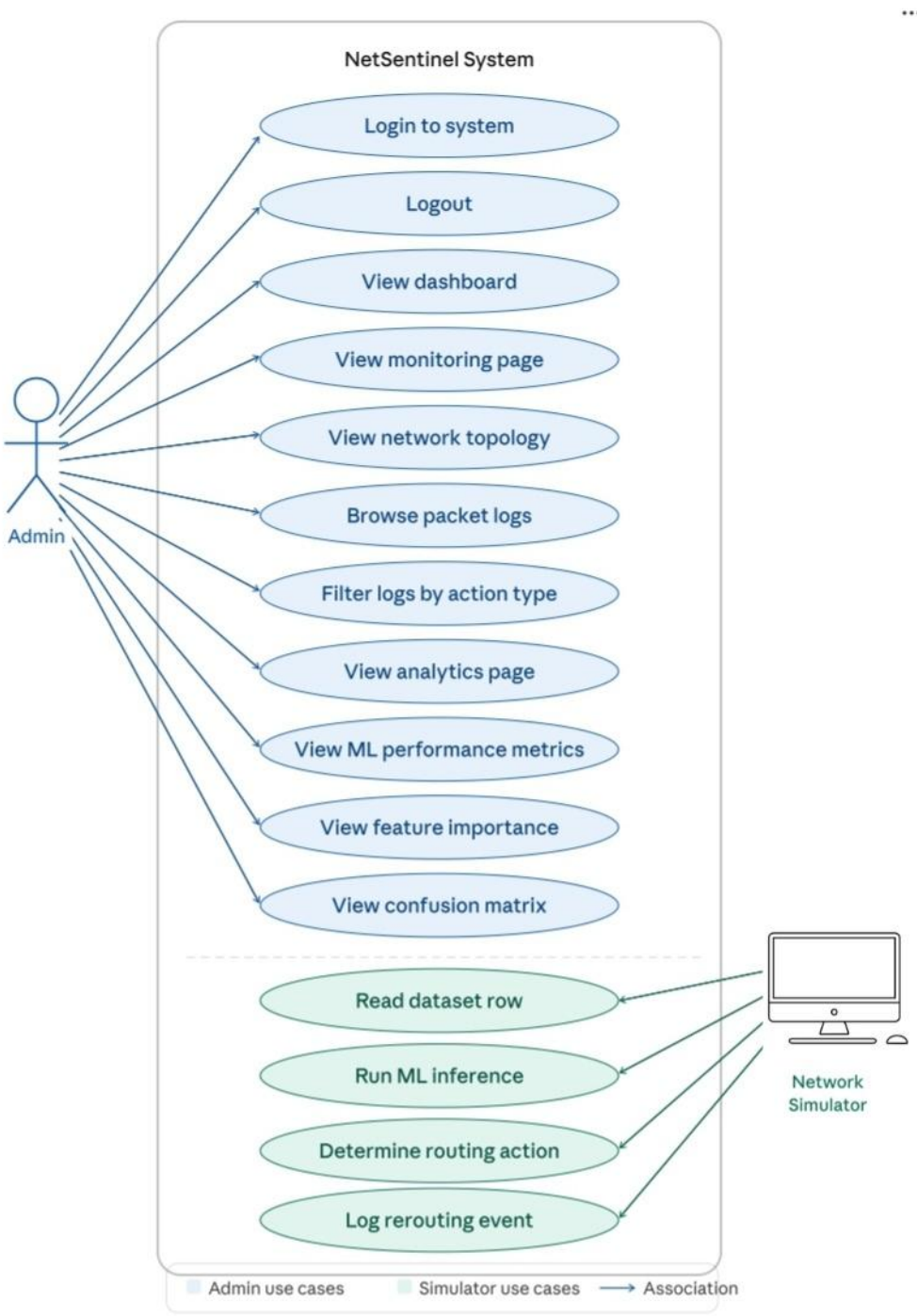

Figure 3.1: Use Case Diagram — NetSentinel System

3.6.2 Activity Diagram — Existing System

The current system flow works as follows: Metric collection using SNMP → Checking threshold via monitoring tool → Breach detection → Manual analysis of alerts by NOC personnel → Manual detection of congestion → Manual rerouting. This diagram represents the current system process, which has been identified to be reactive and dependent on manual intervention.

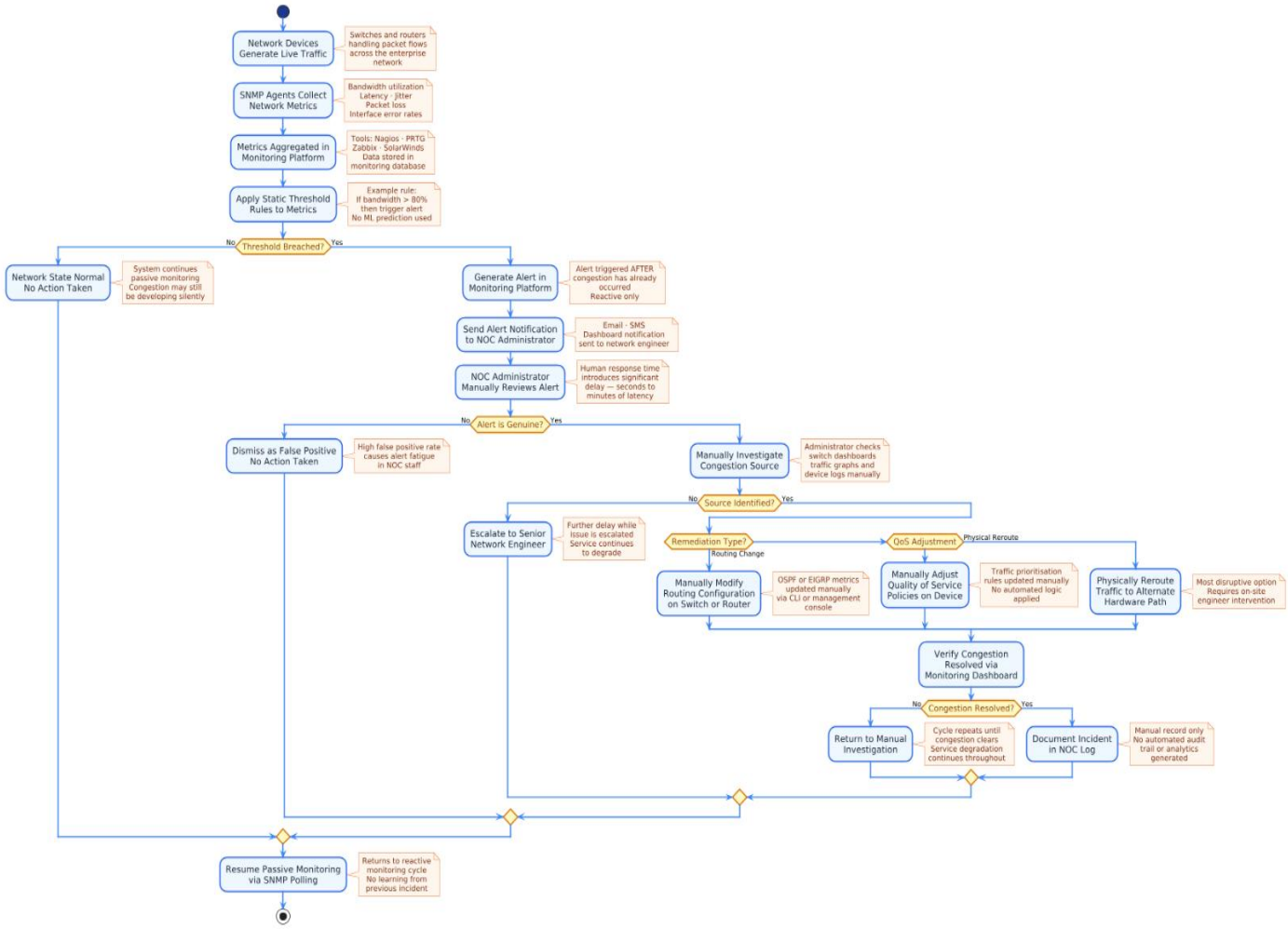

Figure 3.2: Activity Diagram — Existing Rule-Based Network Monitoring System

3.6.3 Activity Diagram — Proposed System (NetSentinel)

The flow of NetSentinel algorithm starts with Network Simulator that reads each row from the dataset and extracts 14 features, followed by an XGBoost congestion model that produces a binary prediction. The prediction is passed to calculate_risk_level function that assigns a

risk level to get_routing_decision function which in turn checks for NORMAL, and sends via WebSocket else, persists a new ReroutedPacket into SQLite and sends to front end.

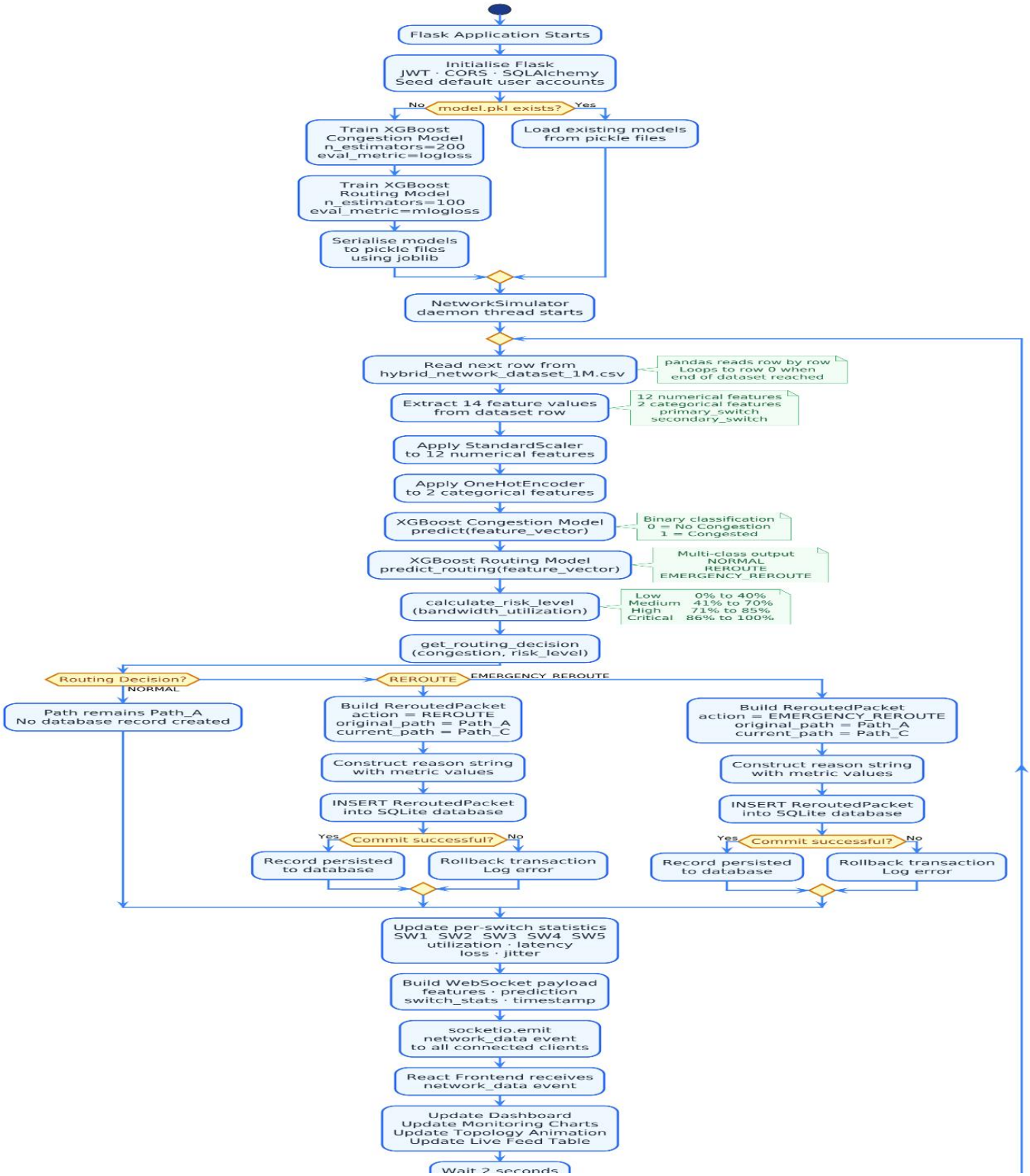

Figure 3.3: Activity Diagram — NetSentinel Congestion Detection and Routing Workflow

3.6.4 System Architecture Diagram

The NetSentinel system is designed based on the three-tiered architecture. The Presentation Layer (React SPA) interacts with the Application Layer (Flask Backend) using Axios HTTP REST requests and Socket.IO WebSocket connections. The Application Layer includes the Flask REST API (consisting of 10 endpoints), ML Engine (two models' inference and routing), and the Network Simulator background process. The Data Layer includes the SQLite database and four model pickle files

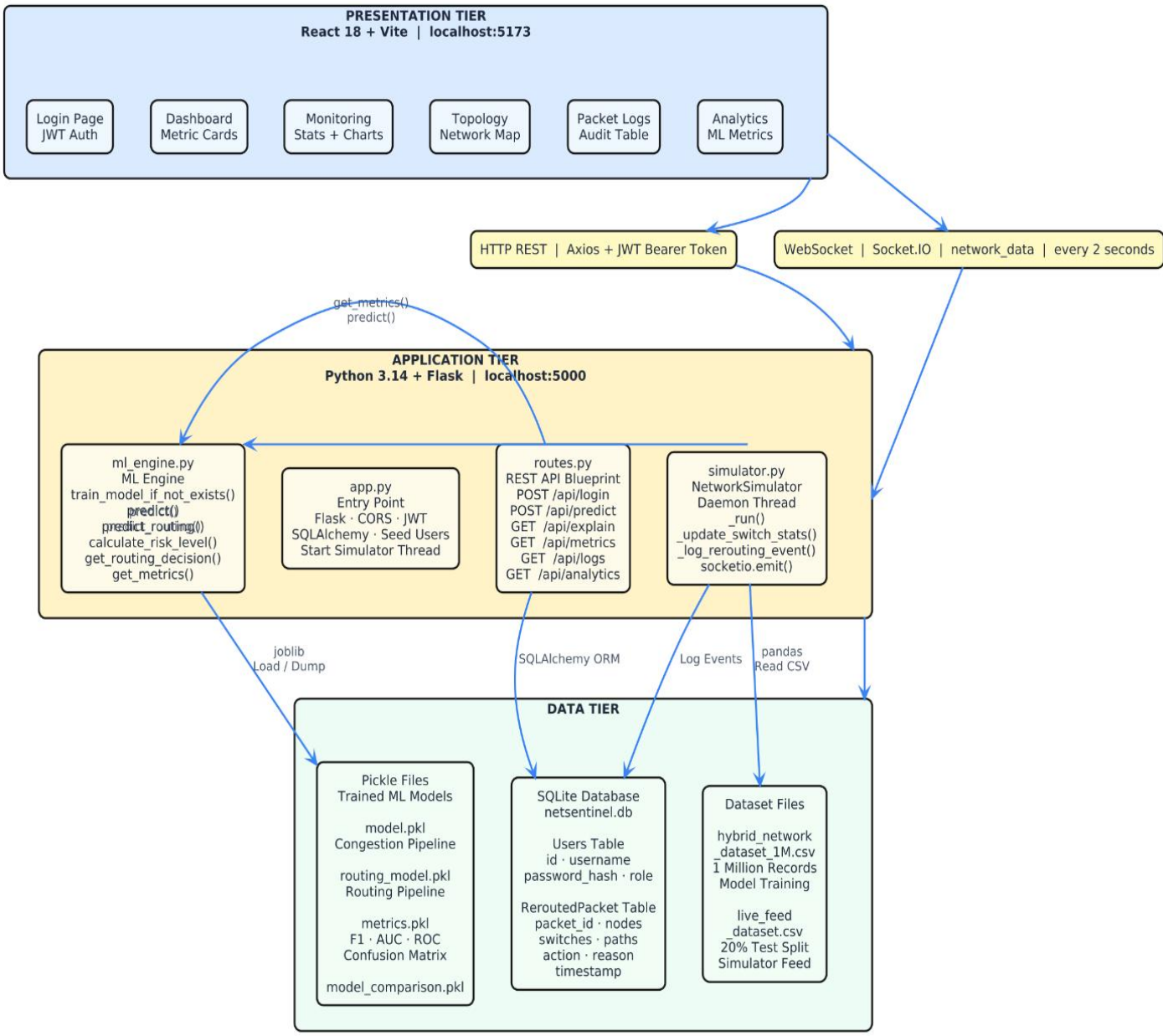

Figure 3.4: Three-Tier System Architecture Diagram — NetSentinel

3.6.5 Class Diagram

Backend Classes: User class (contains id, username, email, password_hash, and role fields along with an authenticate method); ReroutedPacket class (contains id, packet_id, source_node, destination_node, source_switch, destination_switch, original_path, current_path, action, reason, and timestamp fields along with to_dict() and explain() methods); NetworkSimulator class (contains socketio, app, dataset_path, running, and _thread fields along with start(), stop(), and _run() methods); MLEngine module (contains train_model_if_not_exists(), predict(), predict_routing(), calculate_risk_level(), and get_routing_decision() and get_metrics()). APIBlueprint makes use of the User and R

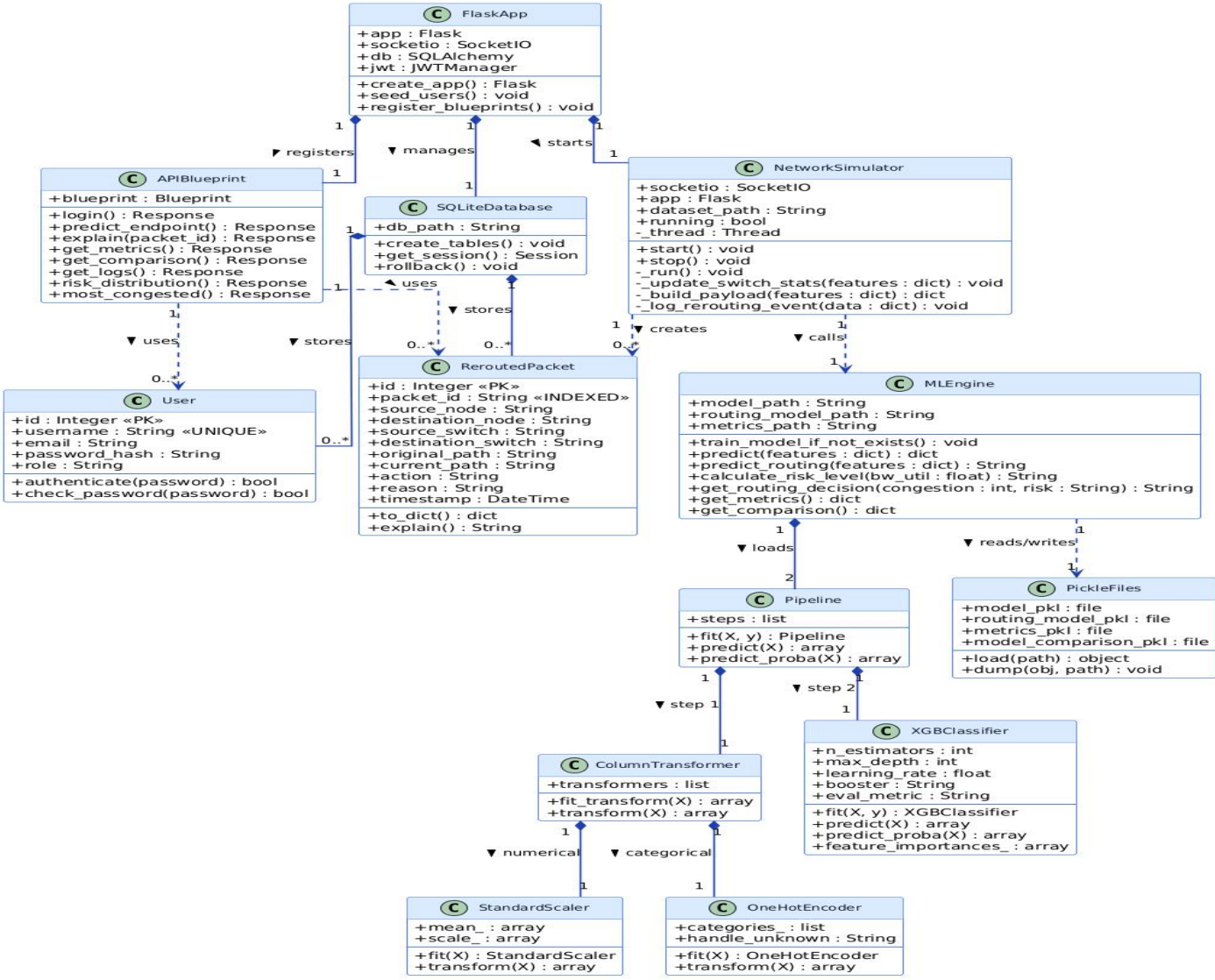

Figure 3.5: Class Diagram — NetSentinel Backend Object-Oriented Design

3.7 Database Design and Schema

The database used is SQLite which is managed by Flask-SQLAlchemy. The reason for using SQLite is because of the embedded and file-based characteristics where there is no need for an external server process. There are two tables in the NetSentinel schema named Users and ReroutedPacket. Two accounts are created at first use: admin and analyst.

Column	Data Type	Constraints	Description
id	INTEGER	PK, AUTOINCREMENT	Unique user identifier
username	VARCHAR(80)	UNIQUE, NOT NULL	Login username — unique at database level
email	VARCHAR(120)	UNIQUE	User email address
password_hash	VARCHAR(256)	NOT NULL	Werkzeug salted hash — never stored as plaintext
role	VARCHAR(20)	NOT NULL	'Admin' or 'Analyst' — enforced by application logic

Table 3.4: Users Table — Column Schema

Column	Data Type	Constraints	Description
id	INTEGER	PK, AUTOINCREMENT	Unique rerouting event identifier
packet_id	VARCHAR(50)	NOT NULL, INDEXED	Unique packet identifier generated by simulator
source_node	VARCHAR(50)	NOT NULL	Source endpoint node (PC1–PC25)
destination_node	VARCHAR(50)	NOT NULL	Destination endpoint node (PC1–PC25)

source_switch	VARCHAR(50)	NOT NULL	Source switch (SW1–SW5)
destination_switch	VARCHAR(50)	NOT NULL	Destination switch (SW1–SW5)
original_path	VARCHAR(100)	NOT NULL	Primary path before rerouting (e.g. Path_A)
current_path	VARCHAR(100)	NOT NULL	Active path after rerouting (e.g. Path_C)
action	VARCHAR(50)	NOT NULL	REROUTE, or EMERGENCY_REROUTE
reason	VARCHAR(512)	NOT NULL	Natural-language explanation of the routing decision
timestamp	DATETIME	DEFAULT CURRENT_TIMESTAMP	Date and time the event was recorded

Table 3.5: ReroutedPacket Table — Column Schema

3.8 System Architecture Design

3.8.1 Backend Architecture

Flask Backend

The Flask backend includes five python files:

app.py (entry point of application - initializes Flask, CORS, JWT, SQLAlchemy, seeds users, starts simulator)

routes.py (ten APIs for authentication, prediction, explanation, metrics, logs, and analytics)

ml_engine.py (training of ML model, inference, risk level computation, route selection, metrics retrieval)

simulator.py (daemon thread that replays dataset, computes statistics per switch, emits events to WebSocket every 2 seconds)

models.py (ORM classes for User and ReroutedPacket)

3.8.2 Frontend Architecture

React 18 frontend is developed using Vite, including six main pages: Login (dark-panel split layout, JWT storage), Dashboard (metric card updates), Monitoring (switch statistic tiles, real-time trend graphs, snapshot panel, live feed table), Topology (routing animation simulation), PacketLogs (audit table filtering), and Analytics (ML visualizations and operations visualization). All requests to the server are placed in services/api.js (Axios + JWT authorization) and Socket connection is handled in services/socket.js (Socket.IO Client).

3.8.3 Risk Level Classification and Routing Logic

Risk Level	Bandwidth Utilization	ML Congestion	Routing Action	Path
Low	0% – 40%	Not congested	NORMAL	Path_A
Medium	41% – 70%	Not congested	NORMAL	Path_A
High	71% – 85%	Congested	REROUTE	Path_C
Critical	86% – 100%	Congested	EMERGENCY_REROUTE	Path_C

Table 3.6: Risk Level Classification Thresholds and Routing Decision Logic

3.9 Evaluation Strategy

Accuracy is an ill-conceived measure when applied to unbalanced classes in a dataset. For the assessment of NetSentinel, however, the following five crucial measures have been employed together with graphical diagnostics on the withheld 20% test data set:

Metric	Formula	Justification
Precision	TP / (TP + FP)	Measures false alarm rate — congestion flags that are not genuine congestion

Recall	TP / (TP + FN)	Measures missed detection rate — a missed congestion event causes unchecked packet loss
F1-Score	2×(P×R)/(P+R)	Primary balanced metric when both false alarms and missed detections are costly
Accuracy	TP + TN / TOTAL	Accuracy measures the proportion of all predictions that are correct both correctly predicted congestion events and correctly predicted non-congestion events combined.
AUC-ROC	Area under ROC curve	Measures discriminatory power across all classification thresholds
Average Precision	Area under PR curve	Summarizes Precision-Recall trade-off; informative when the positive class is of greater concern

Table 3.7: Evaluation Metrics and Justification

Validation utilizes stratified 80-20 hold-out split (random_state=42), maintaining the proportion of congested state label within both splits. Stratified hold-out was selected instead of k-fold cross-validation since this particular dataset is generated using a sequential simulation method, meaning that randomly sampled folds may potentially contain states from the future, thus breaking the time-based consistency. The diagnostic plots generated during validation include the following: confusion matrix, ROC plot, Precision-Recall plot, and feature importance plot – all found within the metrics pickle file and displayed in Analytics.

CHAPTER FOUR
SYSTEM IMPLEMENTATION

4.0 Introduction

This chapter discusses the implementation of the NetSentinel network monitoring and intelligent packet routing system using Artificial Intelligence technology. It begins with an overview of the development environment used to develop the system, and then moves on to discuss the data set used to train the ML model and the architecture and process of training the model. In addition, each component of the system is discussed in detail along with its implementation process. This chapter ends with a discussion on the evaluation of the system, including the results obtained from testing.

4.1 Development Environment and Tools

4.1.1 Programming Language and Libraries

The NetSentinel backend was implemented entirely in Python 3.14, chosen for its extensive ecosystem of data science and web development libraries, its readability, and its widespread adoption in ML-driven application development. The following Python libraries were employed:

a) **Flask (v3.x):** The primary web framework for implementing the REST API. Flask was chosen due to its simplicity and flexibility, along with being compatible with Flask-SocketIO to add WebSocket capability.

b) **Flask-SocketIO:** Enables WebSocket communication between the Flask server and React client using the Socket.IO protocol, facilitating real-time bidirectional data transfer.

c) **Flask-JWT-Extended**: Utilized to provide JSON Web Token based authentication, protecting all API endpoints except the login endpoint which is made public.

d) **Flask-SQLAlchemy:** An ORM integrating SQLAlchemy with Flask for interacting with SQLite database.

e) **Flask-CORS:** Handles the management of Cross-Origin Resource Sharing headers, allowing the React front-end to communicate with the Flask back-end running on a different port.

f) **XGBoost (v2.x):** The primary machine learning library that provides the XGBClassifier for both congestion prediction and routing decision classification.

g) **Scikit-Learn:** Offers implementations of Pipeline, ColumnTransformer, StandardScaler, OneHotEncoder, train_test_split, and other evaluation metrics utilized in the ML pipeline.

h) **Pandas:** Used for dataset handling and feature engineering as well as creating DataFrames for ML inference.

i) **Joblib:** Used to serialize and deserialize pickled ML pipeline object and metrics dictionary objects.

j) **Werkzeug:** Offers methods for generating and checking hashed passwords for user authentication.

The front-end has been developed using React 18 with Vite as the bundler tool. Some important front-end dependencies are as follows:

a. **React (version 18):** Primary UI library using functional components and hooks to manage the application's state.

b. **Axios:** An HTTP library used to make API calls from React components to Flask backend.

c. **Socket.IO Client:** WebSocket client library used to establish a real-time communication link with Flask-SocketIO server.

d. **Lucide-React:** React icon library containing lightweight SVG icons across the dashboard UI.

e. **Tailwind CSS:** Utility-first CSS framework for styling the components used throughout all pages of the dashboard UI.

4.1.2 Development Platform and Hardware Requirements

This was developed on a Windows 11 64-bit machine running with the following configuration: Intel Core i5 @ 2.4 GHz, 8 GB RAM, and 256 GB SSD storage. There was no need for any GPU optimization in terms of both model training and inference, as XGBoost performs well on CPUs.

The development tools employed included VS Code as the main IDE with the use of extensions of Python, ESLint, and Prettier. Backend development was done in a Python virtual environment (.venv). Frontend development was handled using Node.js v22 and npm. Git was used for versioning. The development of the SQLite database utilized DB Browser for SQLite.

Minimum deployment requirements include: Python version 3.10 or higher; Node.js version 18 or above (for the front-end building); 4 GB RAM (8 GB recommended); and 500 MB of free disk space.

4.3 Model Architecture and Training

Two XGBoost classification models were built inside scikit-learn Pipeline components, both including a ColumnTransformer step detailed above.

For congestion predictions, an XGBClassifier is employed with the following hyperparameters: n_estimators=200 (boosting rounds count), max_depth=6 (the maximum depth of each tree), learning_rate=0.1 (shrinkage factor), booster='gbtree' (tree booster type), and eval_metric='logloss' (binary cross-entropy loss for validation purposes).

For routing decision classification, an XGBClassifier with n_estimators=100, max_depth=6, learning_rate=0.1, booster='gbtree', and eval_metric='mlogloss' (logarithmic loss for multi-classification task) is used, since routing decision classification is a four-class problem and was trained on the entire dataset (without splitting into train/test parts) to have all possible classes presented in the training process.

Both models were trained in the train_model_if_not_exists function in ml_engine.py, which was run when the application starts if there are no model pickle files available. Specifically,

training was run only if the absence of model.pkl and model_comparison.pkl was detected, which prevents unnecessary retraining after the server is restarted. Training results (two models and their respective pipelines) are stored in disk in pickle format.

The post-training evaluation of the model on the test set revealed the following evaluation metrics for the congestion prediction model: an accuracy score of 0.9742, a precision score of 0.9718, a recall score of 0.9766, and an F1-Score of 0.9742. The above results indicate that the XGBoost pipeline provides highly accurate performance in terms of congestion detection on the hybrid network dataset.

Accuracy = (TP + TN) / Total = (4,885 + 4,887) / 10,031 = 9,772 / 10,031 = 0.9742 (97.42%)

Precision = TP / (TP + FP) = 4,885 / (4,885 + 142) = 4,885 / 5,027 = 0.9718 (97.18%)

Recall = TP / (TP + FN) = 4,885 / (4,885 + 117) = 4,885 / 5,002 = 0.9766 (97.66%)

F1-Score = 2 × (0.9718 × 0.9766) / (0.9718 + 0.9766) = 0.9742 (97.42%)

4.4 System Implementation and Modules

NetSentinel System Architecture:

NetSentinel system architecture consists of modular application consisting of the following major components:

Backend core: The backend core includes app.py. This file is where the whole application starts by initializing the Flask application and configuring CORS, JWT, and SQLAlchemy. The core of the backend application includes seeding the default users, calling the function that trains the machine learning model, and running the background process NetworkSimulator. The socket events for connecting/disconnecting the client connections are handled here.

Database and models: These components are contained within database.py and models.py respectively. In database.py, the SQLAlchemy engine is initialized, which is later bound to the Flask application in app.py.

REST API (routes.py): The API blueprint defines ten route handlers organized into five functional groups: authentication (/api/login), prediction (/api/predict), explanation (/api/explain/<packet_id>), metrics and comparison (/api/metrics, /api/comparison), packet logs (/api/logs), and analytics (/api/analytics/risk-distribution, /api/analytics/most-congested). All routes except login are protected with the @jwt_required() decorator.

ML Engine (ml_engine.py): This module contains all ML functionalities, including training of the model, preprocessing of features, inference on new input data, calculation of risk level, and decision of routing. The calculate_risk_level and get_routing_decision methods contain logic of the rule-based approach and apply threshold-based risk classification to the ML prediction results. The ML Engine is accessible through clean interfaces of its functions that the API routes and simulator use unaware of internal details of the ML library.

Network Simulator (simulator.py): The NetworkSimulator class controls a simulation process running in background. On each step of simulation, it loads a record from the dataset, performs normalization of feature values, invokes ML inference pipeline and calculation of routing decisions, updates per-switch statistics dictionary (switch_stats), generates detailed reasons for abnormal events, stores information about re-routings in the database, prepares message payload for the WebSocket, and sends it using Flask-SocketIO. The process runs an infinite loop over the entire dataset, starting again from the beginning if it comes to the end.

React Frontend (src/): The front-end consists of six page components: Login, Dashboard, Monitoring, Topology, PacketLogs, and Analytics, along with a common Layout component. All Axios calls are located within the API service file (services/api.js) and use the JWT stored in localStorage as the header for each request. The socket service file (services/socket.js) maintains the Socket.IO connection client. All page components subscribe to their respective REST API endpoints when mounted, while real-time pages (Dashboard, Monitoring, and Topology) subscribe to the 'network_data' Websocket event as well.

Module	Language	Primary Responsibility
app.py	Python / Flask	Application entry point — initialises Flask, CORS,

		JWT, SQLAlchemy; seeds default users; starts NetworkSimulator thread
routes.py	Python / Flask	Ten API endpoints — authentication, prediction, explanation, metrics, logs, and analytics
ml_engine.py	Python / XGBoost	Model training, dual-model inference, risk level calculation, routing decision logic, metric retrieval
simulator.py	Python	Background daemon thread — reads hybrid_network_dataset_1M.csv row by row, runs inference, updates switch stats, logs events, emits WebSocket payload every 2 seconds
models.py	Python / SQLAlchemy	User and ReroutedPacket ORM class definitions with full column schemas
database.py	Python	SQLAlchemy db instance initialisation
Login.jsx	React	Split-panel login page; JWT storage in localStorage; show/hide password; demo accounts display
Dashboard.jsx	React	Real-time summary metric cards; current routing decision and path display; WebSocket subscription
Monitoring.jsx	React	Per-switch statistics tiles; bandwidth/buffer, latency/jitter, packet loss charts; snapshot panel; live feed table
Topology.jsx	React	Animated live network routing topology — SW1-SW5, PC1-PC25, packet path animation
PacketLogs.jsx	React	Searchable, filterable rerouting event audit table with action type filter controls
Analytics.jsx	React	Risk distribution, congestion trend, most

		congested switches, XGBoost performance, model config, feature importance, confusion matrix, ROC curve, Precision-Recall curve

Table 4.1: System Modules and Their Functions

4.5 System Interface Design

The current section highlights the user interface of the implemented NetSentinel application. An illustration of each of the dashboard's six pages is provided together with a brief description. The screenshots are taken from the system running on localhost:5173.

4.5.1 Login Page

The Login view utilizes the split-pane dark theme having the NetSentinel branding pane on the left side which consists of the stats pane (Accuracy: 97.4%, +97.2% Precision, 2-model pipeline) whereas the authentication pane is on the right containing the username and password input controls with password visibility switch, Sign In button and collapsible demo account credentials in a mono-spaced font. After successful authentication, the JWT token is stored locally and the user is redirected to the Dashboard view.

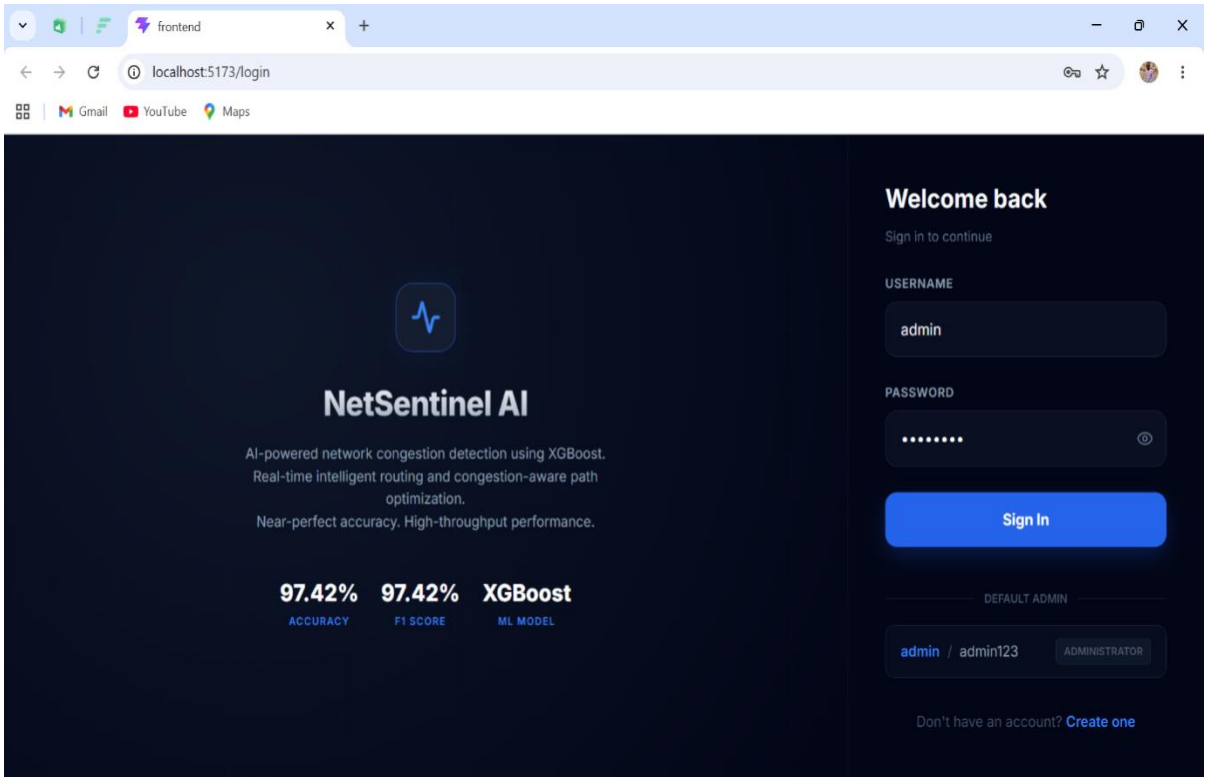

Figure 4.1: Login Page

4.5.2 Dashboard Page

The Dashboard is the first landing page after the user has authenticated. It has summary metric cards for all of the following metrics at runtime: Latency, Jitter, Throughput, Bandwidth Utilization, Buffer Occupancy, Packet Loss Rate, and Risk Level. The Routing Decision and Active Path will be shown prominently on the dashboard and will refresh automatically every time there is an update received from the NetworkSimulator via WebSockets every two seconds.

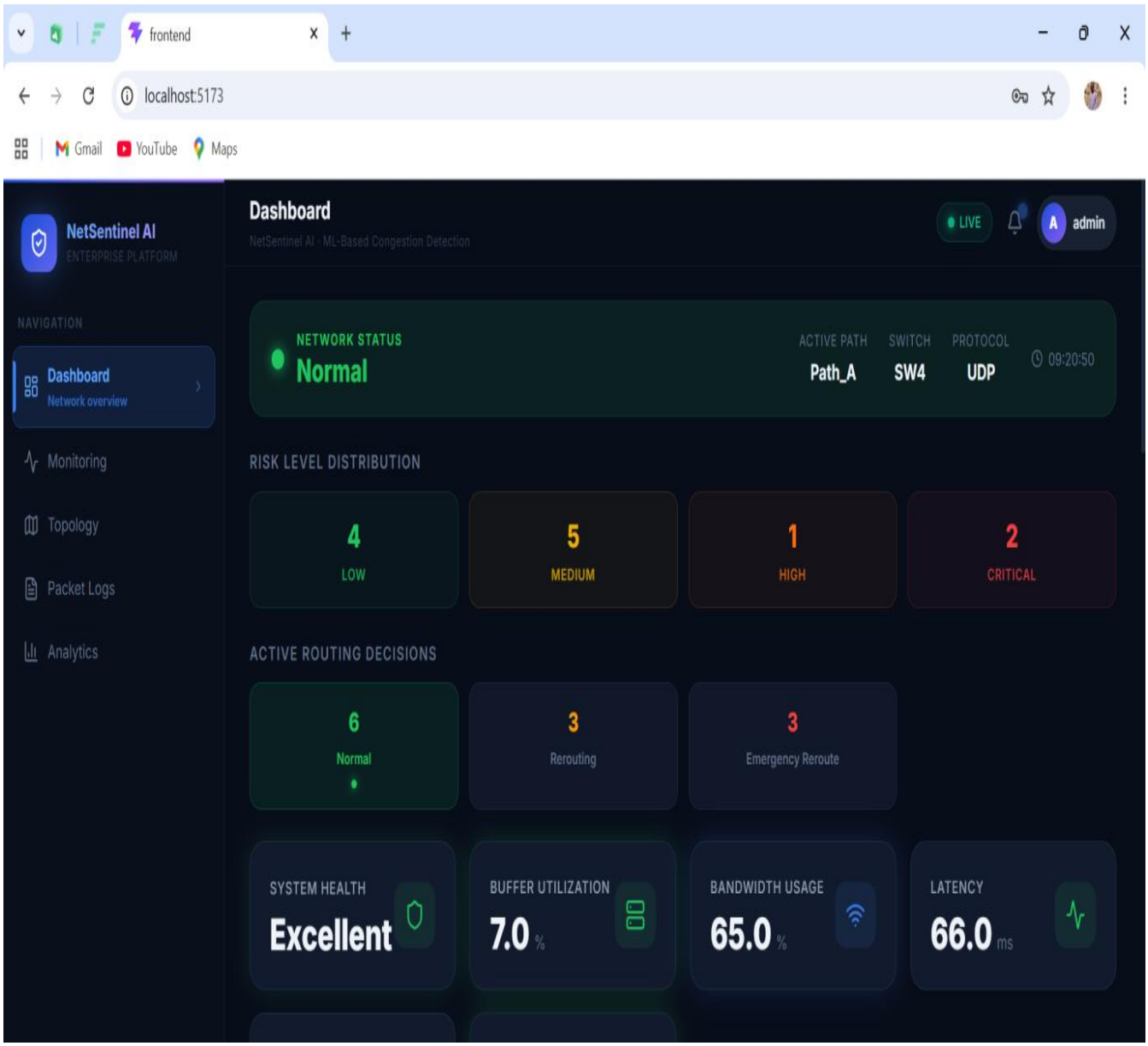

Figure 4.2: Dashboard Page

4.5.3 Monitoring Page

The Monitoring page offers the most detailed real-time data on the network performance. At the top, there are five per-switch statistic cards (SW1 through SW5), each card indicating utilisation percentage, latency, packet loss rate, and jitter, and using four color indicators (Normal – green, Warning – yellow, High – orange, Critical – red). The next part features three real-time line graphs – Bandwidth & Buffer (%), Latency & Jitter (ms), and Packet Loss (%), each containing forty most recent entries. There is also the Current Snapshot panel

displaying the latest values for each metric. On the last page, you can see the Live Network Feed Table containing individual packet data with the following parameters – TIME, SRC IP, DST IP, PROTO, PATH, LATENCY, BW%, BUFFER%, LOSS%, JITTER, RISK, and DECISION.

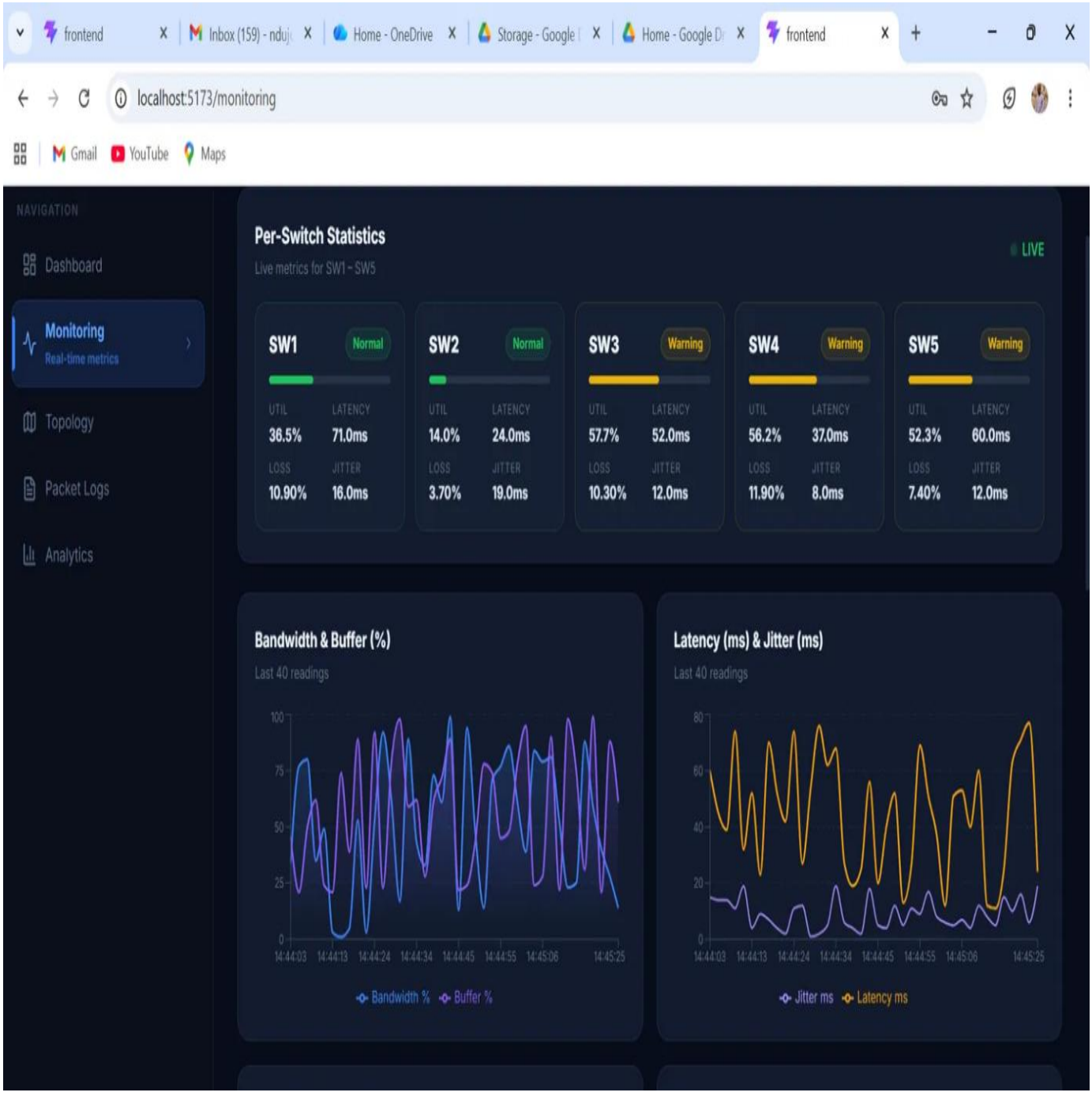

Figure 4.3: Monitoring Page(Per-Switch Statistics with Status Badges)

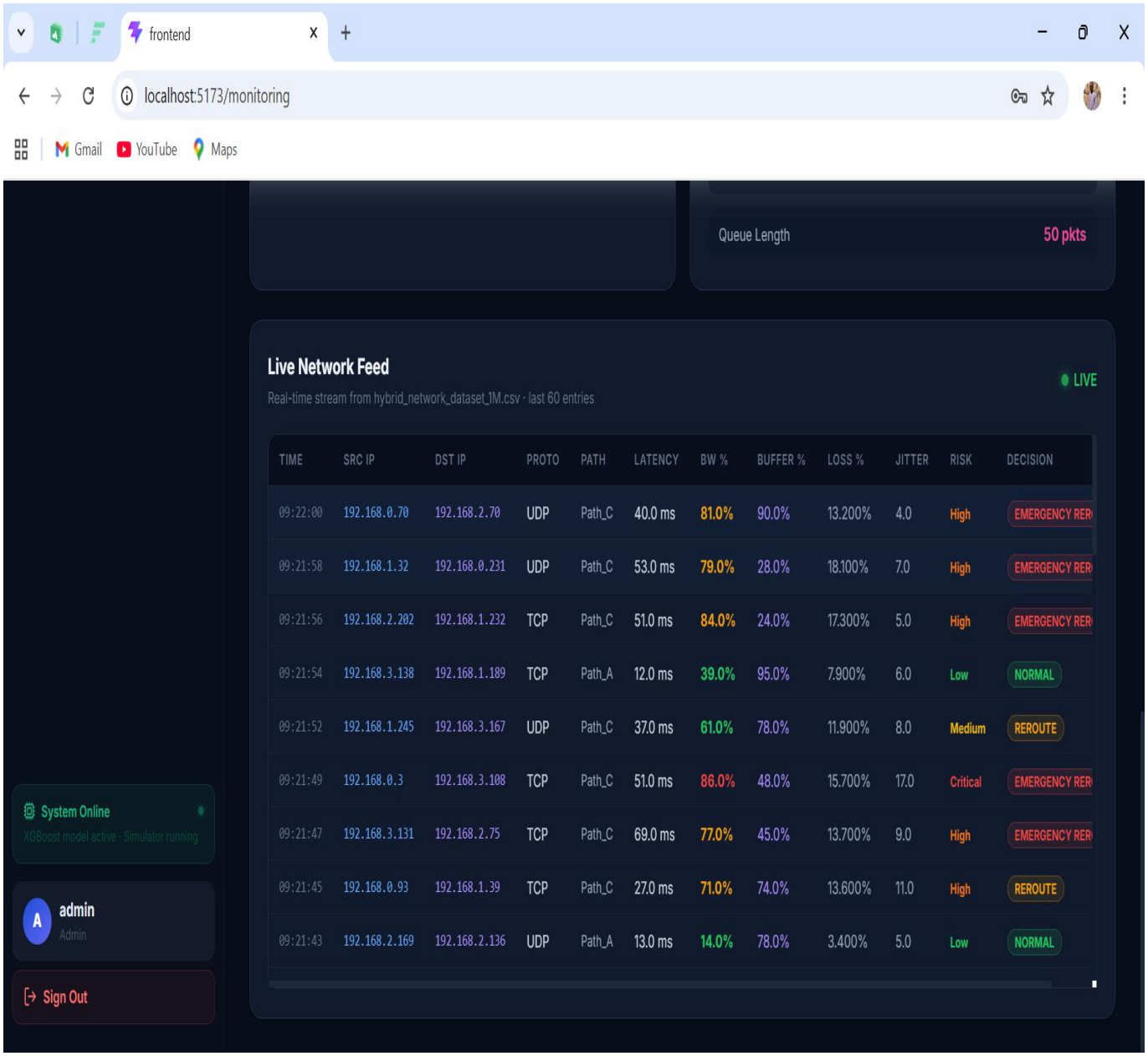

Figure 4.4: Monitoring Page (Live Trend Charts and Network Feed Table)

4.5.4 Topology Page

In the Topology tab, an animation of a live network routing simulation canvas is shown. In the canvas, there are five switch nodes that are represented as larger circles with labels such as SW1 to SW5. In addition, there are twenty-five endpoint nodes represented as smaller nodes and located around the switches, labeled PC1 to PC25. The animated routes taken by packets are color-highlighted in two colors. Green lines are used to represent Packet Path_A

while orange lines indicate Packet Path_C. The source and destination nodes are marked as SOURCE and DEST.

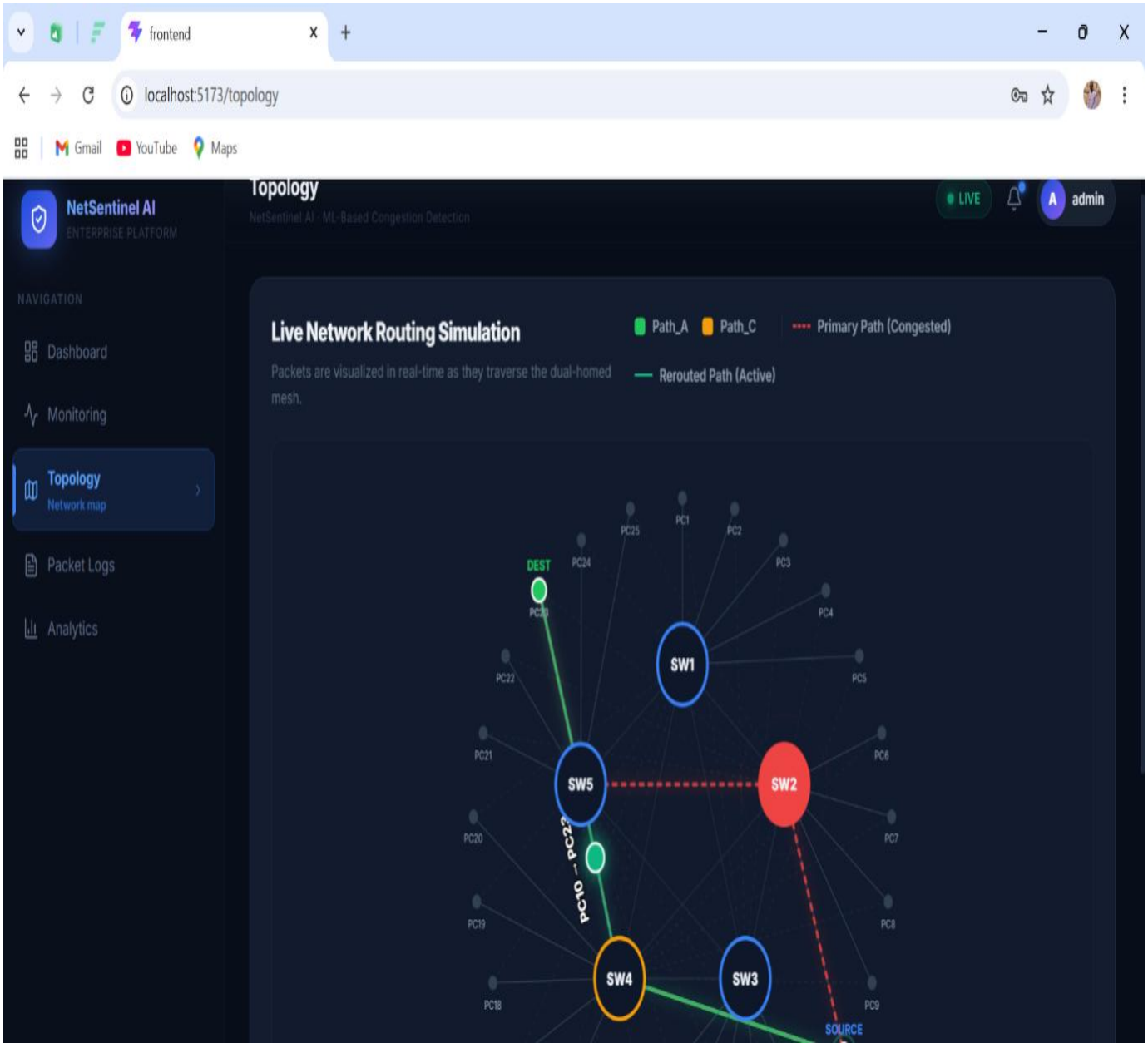

Figure 4.5: Topology Page — Live Animated Network Routing Simulation

4.5.5 Packet Logs Page

The Packet Logs page is an exhaustive audit table where all the NON NORMAL rerouting events from the SQLite database can be seen. Each entry includes all details about the ReroutedPacket record such as packet_id, Source node, Destination node, Source switch,

Destination switch, Original path, New path, Routing action, Reason for routing, and Date and Time of the event. The data can be filtered based on the action (REROUTE or EMERGENCY_REROUTE) with the help of filters.

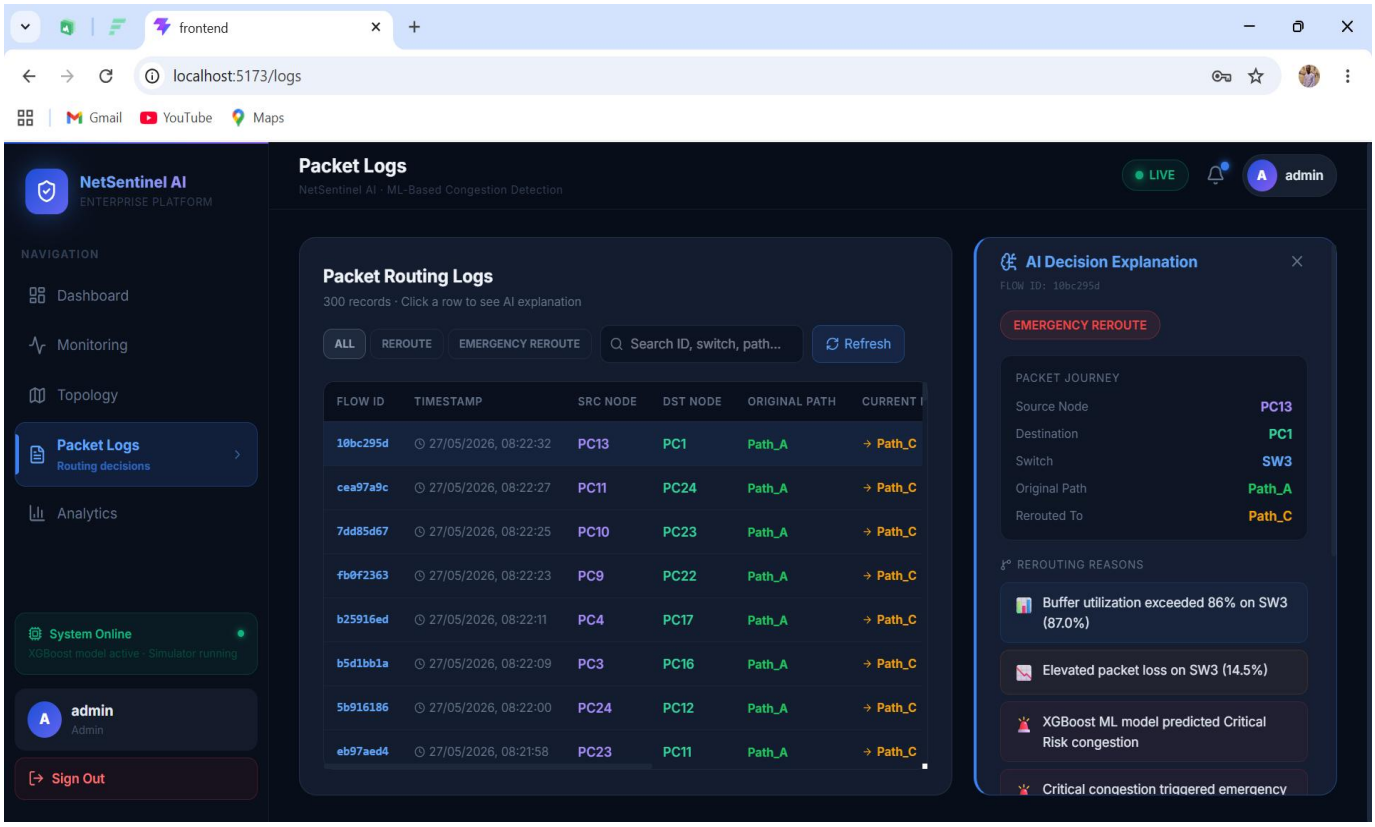

Figure 4.6: Packet Logs Page — Complete Rerouting Event Audit Trail

4.5.6 Analytics Page

The Analytics page represents the most extensive visualisation page of the entire system. This page has been split into two parts. The upper part displays operational analytics: four gauges indicating the accuracy (97.4%), precision (97.2%), recall (97.7%), and F1-score (97.4%) of the model; an Active for Inference badge; a donut chart representing the risk level distribution; a line chart that displays the congestion trend over time and a threshold at 70% bandwidth; a most congested switches bar chart with rankings from SW1 to SW5 according to the number of rerouting events; and finally, a reference table with the model configuration and the routing logic. The lower part contains ML explainability visualisations: the feature importance chart of the XGBoost model, the confusion matrix, the ROC curve, and the Precision-Recall curve.

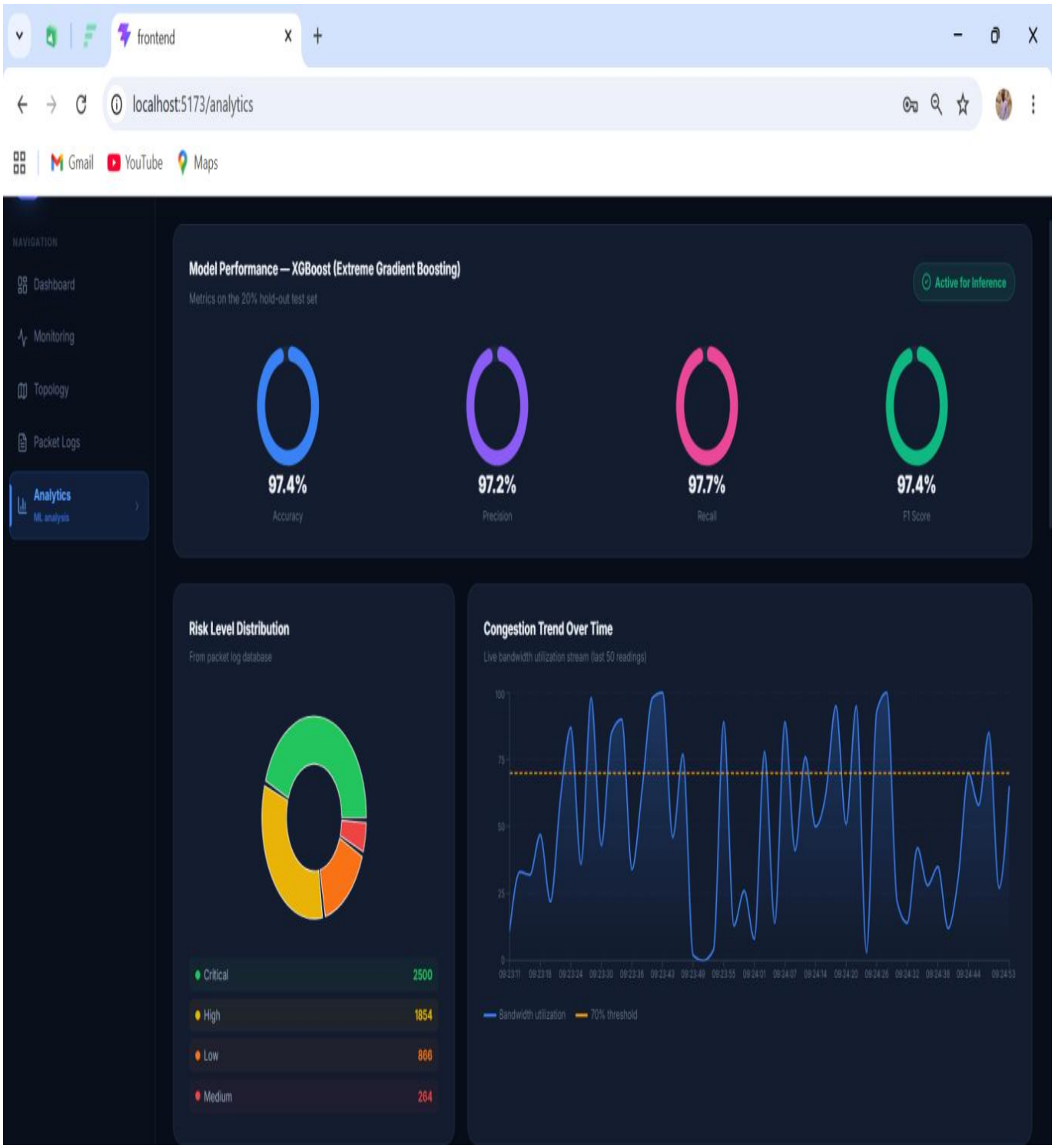

Figure 4.7: Analytics Page — Model Performance Gauges and Risk Distribution

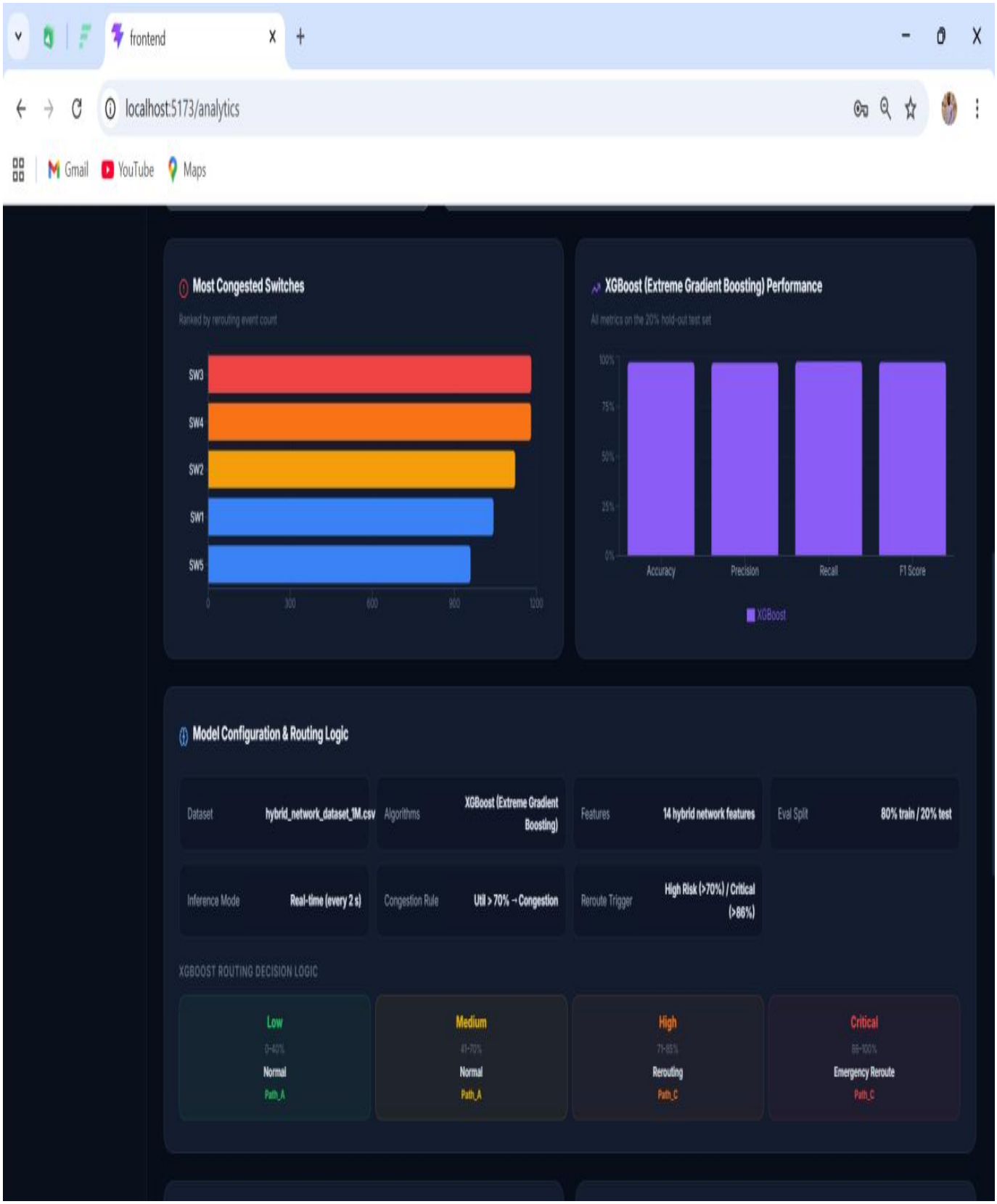

Figure 4.8: Analytics Page — Most Congested Switches and XGBoost Performance Chart

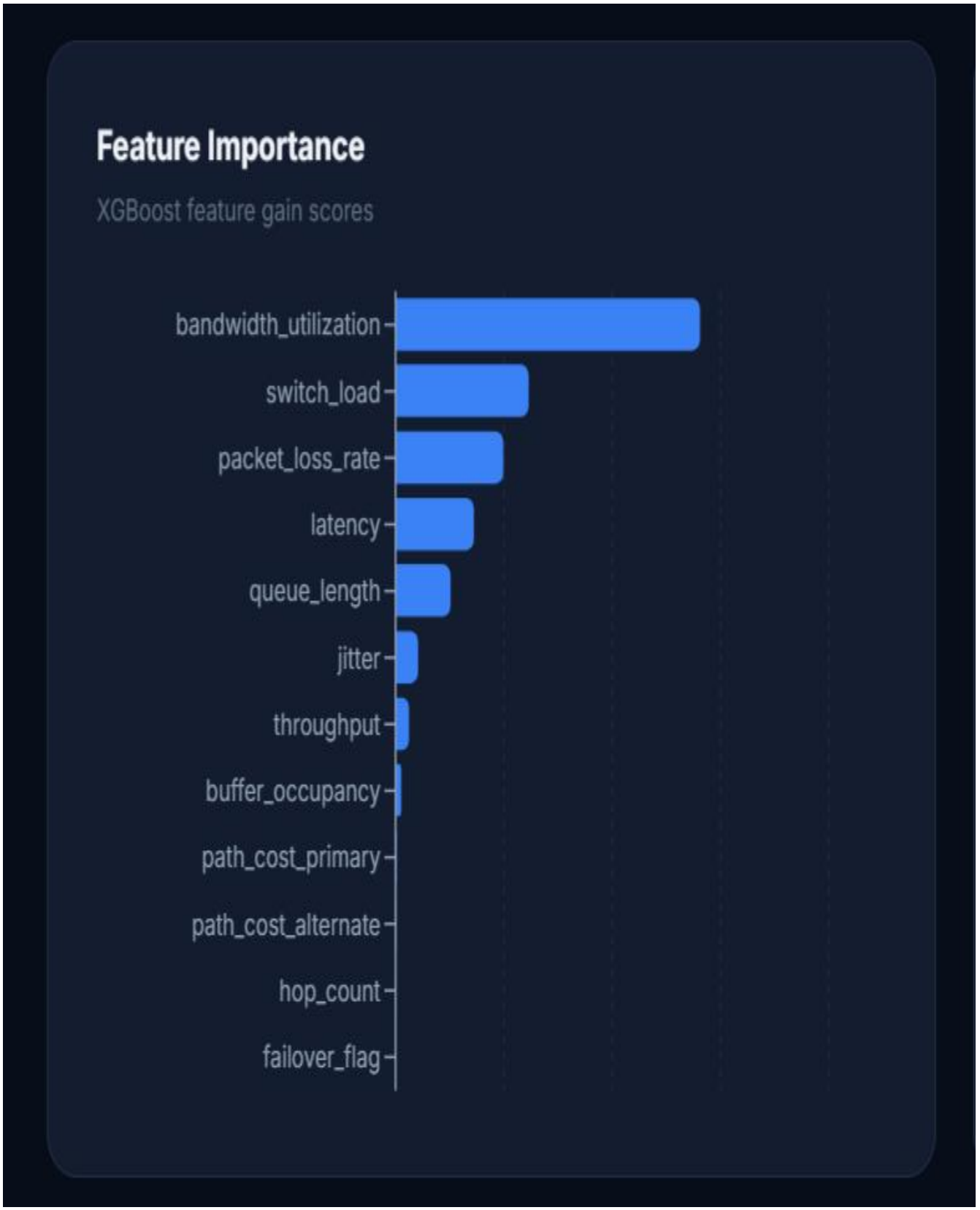

Figure 4.9: Analytics Page — XGBoost Feature Importance Chart

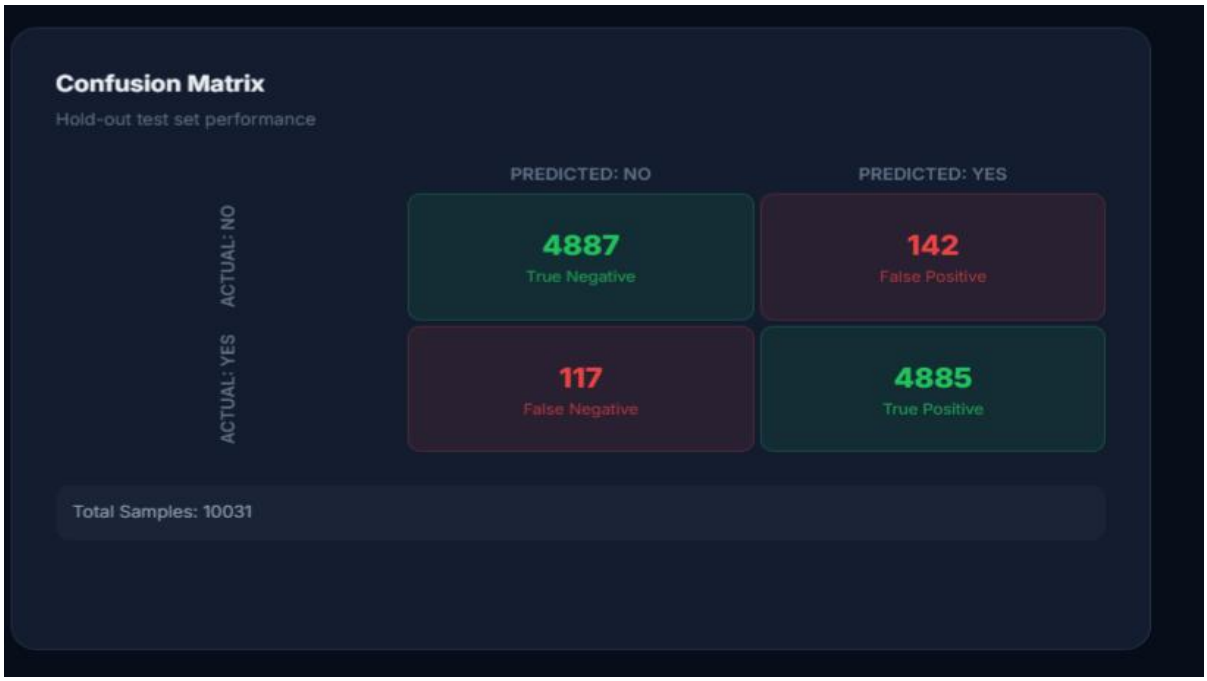

Figure 4.10: Analytics Page — Confusion Matrix (Hold-Out Test Set Performance)

Figure 4.11:Analytics Page — ROC Curve (AUC=0.9885) and Precision-Recall Curve (AP=0.9854)

4.5 Testing and Evaluation

4.5.1 Evaluation Metrics

The performance of the ML algorithm was analyzed based on four common classification evaluation metrics calculated for the 20% holdout dataset. Accuracy is the ratio of correct

classifications against the total number of classifications. Precision denotes the accuracy of positive (congested) predictions relative to all positive predictions, which demonstrates system immunity to false-positive results. Recall, also known as sensitivity, stands for the accuracy of detecting congestions out of all occurrences of congestion in the test data. F1 Score is the harmonic mean of precision and recall and gives an even-weighted value for precision and recall together. The confusion matrix was also computed.

4.5.2 System Testing

Testing was done at three different stages - unit testing of the backend modules, integration testing of the APIs, and end-to-end testing of the entire pipeline.

Unit Testing: The ML engine functions (calculate_risk_level, get_routing_decision, predict, predict_routing) were tested with representative input values representing all the risk level boundary values and routing decision scenarios. The database models were tested by performing Create, Read, Update, and Delete operations with respect to the User model and the ReroutedPacket model.

Integration Testing: All the ten API endpoints were tested using the HTTP client called Postman. Test cases covered issuance of JWT on valid login, return of status code 401 when no token or invalid token is passed, 400 errors on absence of feature predictions, correctness of prediction output values, and correctness of the JSON response format for the analytics endpoint.

End-to-End Testing: End-to-end testing was conducted for the entire system, whereby both the frontend and backend operated simultaneously, and the monitoring dashboard was monitored for five minutes to check whether data transmission using the WebSocket was successful, risk levels were displayed correctly, the rerouting event was triggered successfully, logged, and the topology and analytics views rendered appropriately. There were no crashes, memory leaks, or inconsistencies during this process.

4.5.3 User Interface Testing and Walkthrough

- Interface testing was conducted to ensure that all dashboard pages worked properly in Google Chrome (the latest version), Microsoft Edge, and Mozilla Firefox. The pages tested include the following:
- Login Page: The login page successfully checks for empty fields, displays an error message if the entered credentials are wrong, and redirects to the Dashboard page if login is successful.
- Dashboard Page: This page shows real-time summary cards for Latency, Jitter, Throughput, Utilization, Buffer Occupancy, Packet Loss, and Risk Level. Routing Decision and Path change accordingly with every WebSocket event.
- Monitoring Page: This page shows real-time feed for Network Events with additional information about each packet. Information includes Source/Destination Nodes, Switch ID, Congestion Level, Routing Decision, and Risk Level. The page has color coding for different levels of alertness: NORMAL (green), REROUTE (orange), and EMERGENCY_REROUTE (red).
- Topology Page: This page visualizes five switches (SW1-SW5) and connection between them. Additionally, each switch is accompanied by its current Utilization percentage, as well as Status (normal, warning, high, critical).
- Packet Logs Page: The packet logs page renders the re-routing history properly with all fields. Filtering works perfectly by filtering based on action types. The table can be scrolled easily for viewing many entries.
- Analytics Page: The risk distribution graph, list of most congested switches, and model metrics table all render properly with data retrieved from REST API endpoints.

4.6 Results Presentation and Analysis

4.6.1 Output Samples and Interface Screens

There are different types of outputs generated by the software that demonstrate its effectiveness. In the monitoring dashboard, there is a sample event payload showing the following information: primary_switch: SW3, bandwidth usage: 88.4%, buffer usage: 91.2%,

latency: 142.7 ms, jitter: 18.3 ms, packet_loss_rate: 0.082, risk_level: Critical, routing_action: EMERGENCY_REROUTE, route: Path_A, current_route: Path_C. From this event payload, one can see how effective the software is since it identifies critical congestion at SW3, resulting in an emergency re-routing action.

In the packet log tab, there is an audit trail of all the non-NORMAL routing decisions kept in the SQLite database. One such representative entry is as follows: Packet ID: a4c7f1b2, Source Node: PC11, Destination Node: PC24, Source Switch: SW3, Destination Switch: SW5, Route: Path_A, current Route: Path_C, Action: EMERGENCY_REROUTE, Reason: Buffer utilization reached 86% at SW3 (88.4%) | Latency high on path-A (142.7 ms) | XGBoost Machine Learning model predicts Critical Risk congestion | Critical congestion led to emergency re-routing, Timestamp: 202

4.6.2 Discussion of Results and System Performance

The performance results of the evaluation conducted for the system suggest that the combination of the XGBoost-based machine learning solution with the real-time monitoring and routing system works as intended to fulfill the goals of the research project.

The accuracy and F1 score of the congestion prediction model are at 97.42%, indicating that the XGBoost approach reliably classifies data into either congested or non-congested states based on the fourteen input variables provided. The high recall score of 97.66% can be especially emphasized since false negatives (non-identification of congestion) will have more adverse consequences in such application than false positives due to the nature of the problem addressed.

Regarding routing decisions, the combined strategy of congestion prediction and threshold-based rules was able to classify states as either EMERGENCY_REROUTE, REROUTE, or NORMAL, depending on the risk level. The former category corresponds to congestion levels above 86% utilization, while the latter two correspond to lower-risk conditions.

CHAPTER FIVE
SUMMARY , CONCLUSION , RECOMMENDATIONS

5.1 Introduction

The current chapter is one that marks the end of the current research by giving an overall summary of what the researcher has done, highlighting conclusions in relation to meeting the objective, and also giving recommendations on how further research can be done and suggestions for improvements in the system. This chapter is meant to give a reflection of the entire process involved starting from defining the problem up to the evaluation process.

5.2 Summary of the Study

The purpose of this research was to tackle the issues that exist in relation to traditional and rule-based approaches to network monitoring and routing through the creation of NetSentinel, an AI-driven network monitoring and intelligent packet routing solution. The reasons behind carrying out this research included the increasing ineffectiveness of a response-based and threshold-oriented approach to network management as well as the potential of machine learning.

5.3 Conclusion

The purpose of this research was to design and develop a network management system using intelligent systems through machine learning algorithms to predict network congestion and optimize the process of routing. This objective has been met according to the outcomes of Chapter Four.

5.4 Recommendations

- The following recommendations can be made based on the findings from this study and the lessons learned from developing the system solution:
- •Network Live Data Collection: In future studies, it is recommended that the The system be connected to the live networks through a packet capture module using Python-based packet capture libraries like Scapy and PyShark, which will allow the

system to operate on live data instead of simulated datasets, necessitating retraining of the machine learning model on the specific traffic.

- •Integration with SDN Controllers: In order to implement the automated routing decision process, the routing decisions made by the machine learning algorithm must be integrated with an SDN controller like OpenDaylight or ONOS, which would allow for the implementation of the routing decision in the form of flow rules on physical/virtual OpenFlow switches.
- Mobile App:There is a need to design a companion mobile application for use in both Android and iOS systems that will notify network administrators of any critical risk event and EMERGENCY_REROUTE in real time.

REFERENCES

Amarasinghe, K., Kenney, K., & Manic, M. (2018). Toward explainable deep neural network based anomaly detection. Proceedings of the 11th International Conference on Human System Interaction (HSI) (pp. 311-317). IEEE.

Azzouni, A., & Pujolle, G. (2017). NeuRoute: Predictive dynamic routing for software-defined networks. Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM) (pp. 331-338). IEEE.

Benson, T., Akella, A., & Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC) (pp. 267-280). ACM.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., & Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: Evolution, applications, and research challenges. Journal of Internet Services and Applications, 9(1), 16.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). CRISP-DM 1.0: Step-by-step data mining guide. SPSS Inc.

Chen, T., & Guestrin, C. (2016). XGBoost:

Fadlullah, Z. M., Tang, F., Mao, B., Kato, N., Akashi, O., Inoue, T., & Mizutani, K. (2017). State-of-the-art deep learning: Evolving machine intelligence towards tomorrow's intelligent network traffic control systems. IEEE Communications Surveys and Tutorials, 19(4), 2432-2455.

Hui, H., Bao, W., Yuan, D., Deng, S., Zhou, A., & Zhang, J. (2021). Collaborative online learning scheme for 5G heterogeneous networks: A deep reinforcement learning approach. IEEE Transactions on Network and Service Management, 18(4), 4367-4380.

Jain, S., Chaudhari, N. S., & Gulhane, A. (2022). Comparative analysis of explainability techniques for XGBoost-based network intrusion detection. Journal of Ambient Intelligence and Humanized Computing, 13(7), 3469-3481.

Kato, N., Fadlullah, Z. M., Mao, B., Tang, F., Akashi, O., Inoue, T., & Mizutani, K. (2017). The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective. IEEE Wireless Communications, 24(3), 146-153.

Liu, Y., Zhang, L., Guo, M., & Chen, J. (2019). A machine learning-based QoS routing scheme for multi-service software-defined networks. Journal of Network and Computer Applications, 143, 1-10.

Mao, H., Netravali, R., & Alizadeh, M

Nguyen, T. T. T., & Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. IEEE Communications Surveys and Tutorials, 10(4), 56-76.

Shi, W., Li, N., Wu, J., & Cao, X. (2021). Intelligent traffic management framework using XGBoost for congestion detection in vehicular ad hoc networks. IEEE Transactions on Vehicular Technology, 70(9), 8963-8977.

Stampa, G., Arias, M., Sanchez-Charles, D., Muntes-Mulero, V., & Cabellos, A. (2017). A deep-reinforcement learning approach for software-defined networking with OpenAI Gym. arXiv preprint arXiv:1709.07080.

Suresh, L., Schulz-Zander, J., Merz, R., Feldmann, A., & Vazao, T. (2017). Towards programmable enterprise WLANs with Odin. Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN).

Xiao, Y., Xiong, M., & Wang, H. (2019). An ensemble learning approach for intelligent routing in software defined networks. Proceedings of the 2019 IEEE International Conference on Communications (ICC), 1-6.

Xie, J., Yu, F. R., Huang, T., Xie, R., Liu, J., Wang, C., and Liu, Y. (2018). A survey on machine learning techniques for software-defined networking (SDN): Research issues and challenges. IEEE Communications Surveys and Tutorials

Zhang, C., Patras, P., and Haddadi, H. (2018). Deep learning for mobile and wireless networking: A survey. IEEE Communications Surveys and Tutorials, 21(3), 2224-