A HYBRID MODEL FOR IMPROVING THE EFFICIENCY OF FOG LAYER IN THE INTERNET OF THINGS

Watson Musyoki Kanuku

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Information Technology of Murang'a University of Technology

August, 2026

DECLARATION

Student Declaration

I hereby declare that this thesis is my original work and to the best of my knowledge has not been presented for a degree in this or any other University. I understand what Plagiarism is and I am aware of the University policy in this regard. I understand that any false claim in respect of this work shall result in disciplinary action in accordance with the University Anti-Plagiarism Policy.

_____ _____

Watson Musyoki Kanuku Date

SC500/5564/2021

Supervisors' Approval

The undersigned certify that they have read and hereby recommend for acceptance of Murang'a University of Technology a proposal entitled "**A Hybrid Model for Improving the Efficiency of Fog Layer in The Internet of Things".**

_____ _____

Dr. Stephen Njenga Date

Department of Computer Science

Murang'a University of Technology

_____ _____

Dr. George Musumba Date

Department of Computer Science

Dedan Kimathi University of Technology

DEDICATION

I dedicate this thesis to my parents, my late father Daniel Kanuku Malinda who immensely valued education, and inspired me to strive to reach the highest level of education. To my mother who despite never having been to a formal school, never spared her efforts to jointly educate their 11 children amidst a myriad of challenges.

ACKNOWLEDGEMENT

I give all glory and honor to Almighty God for granting me the mental strength, financial provision, and spiritual support that enabled me to successfully undertake this research and complete this thesis. His grace has sustained me throughout this journey.

I deeply appreciate my supervisors, Dr. Stephen Njenga and Dr. George Musumba, for their dedication, patience, and invaluable guidance throughout this research process. Their mentorship greatly shaped this work. I am also thankful to Dr. John Ndia for his support in implementing the corrections recommended by the Board of Examiners. My gratitude goes to my classmates and friends for their encouragement, insightful discussions, and support in refining my research ideas.

I extend my sincere appreciation to Murang'a University of Technology management for initiating this program, which has provided me with the opportunity to pursue and achieve my academic aspirations. I am equally grateful to the School of Computing and Information Technology for effectively facilitating the coursework and providing the necessary academic resources. Special thanks go to the Department of Information Technology for their unwavering support, ensuring smooth delivery of classes and a conducive learning environment. I also acknowledge the Board of Postgraduate Studies for their guidance and commitment to maintaining the quality and standards of this thesis.

Finally, I am profoundly grateful to my family my wife Joan, my son Isaac, and my daughters Sophie and Pearl for their unwavering love, encouragement, and prayers. Whenever my strength waned, they inspired me to keep going. It is through God's grace and their support that I have been able to achieve this important academic milestone.

ABSTRACT

Fog computing relies on many interconnected, decentralized devices working together to process fast moving and diverse data. The rapid growth and complexity of this data, in terms of volume, speed, variety, and variability, often slow down data transfer and processing. This delay affects real-time IoT applications like smart healthcare and autonomous vehicles. Despite the importance of responsive data handling, fog layer efficiency has not received enough attention. With more edge devices coming online, there's an urgent need for faster data processing and response. This study developed a hybrid machine learning model combining DBSCAN for clustering, and KNN for classification, to improve fog computing efficiency. A mixed methods approach grounded in pragmatic research philosophy was used. First, a systematic literature review identified key variables influencing fog efficiency. These were ranked using a Random Forest-Recursive Feature Elimination (RF-RFE) algorithm. Input from IoT experts selected purposively was sought through a survey. Snowball sampling was used to identify 30 respondents to validate the ranking. Key variables included Packet Loss Rate, Queue Time, Latency, Channel Utilization, and Throughput. Using a Design Science Research framework, the hybrid DBSCAN+KNN model was developed and tested using the YAFS simulator. The hybrid model outperformed the standard KNN algorithm, achieving 95.33% accuracy, compared to KNN's 93.67%. It also reached a recall of 99%, surpassing both classic KNN 95.2% and ensemble KNN 89.68% from previous studies. These results show that DBSCAN effectively pre-processes data by grouping similar values and reducing noise before classification by KNN, leading to faster and more accurate processing. This hybrid approach offers a promising way to boost fog efficiency in time-sensitive IoT environments, unlocking the full benefits of smart devices.

TABLE OF CONTENTS

DECLARATION ..... **ii**
DEDICATION ..... **iii**
ACKNOWLEDGEMENT ..... **iv**
ABSTRACT ..... **v**
LIST OF TABLES ..... **x**
LIST OF FIGURES ..... **xi**
ACRONYMS AND ABBREVIATIONS ..... **xii**
CHAPTER ONE: INTRODUCTION **1**
1.1 Background of the study 1
1.2 Problem Statement 4
1.3 Objectives 5
1.3.1 Main Objective 5
1.3.2 Specific Objectives 5
1.4 Research Questions 5
1.5 Scope of the Study 6
1.6 Significance of the Study 6
1.7 Contributions of the Thesis 7
1.7.1 Contribution to Theory 7
1.7.2 Contribution to practice 8
1.8 Organization of the Thesis 8
CHAPTER TWO: LITERATURE REVIEW **9**
2.1 Introduction 9
2.2 The IOT Architecture 9
2.2.1 Thing Layer 10
2.2.2 Fog Layer 11
2.2.3 Cloud Layer 11
2.3 Theoretical Foundations 12
2.4 Efficiency of the Fog Layer 14
2.5 Data Processing in the Fog Computing 15
2.6 Hybrid Approaches in IOT 25
2.7 Clustering Algorithms 27
2.8 Classification Algorithms 29
2.9 Fog Simulation Platforms 33
2.10 Paradigm of Synthetic Dataset 35
2.10.1 Synthetic Dataset in Machine Learning 35
2.10.2 Case Studies on Synthetic Dataset 35
2.11 Variables Determining Fog Efficiency 37

2.12.1 Latency 37
2.12.2 Chanel Utilization 39
2.12.3 Throughput 40
2.12.4 Response Time 41
2.12.5 Packet Loss Rate 43
2.12.6 Queue Time 44
2.13 Research Gaps 46
2.14 Conceptual Framework 47
2.15 Chapter Summary 48
CHAPTER THREE: METHODOLOGY 49
3.1 Introduction 49
3.2 Research Philosophy 49
3.3 Research Design 51
3.3.1 Research Design for Objective One 53
3.3.2 Research Design for Objective Two 53
3.3.3 Research Design for Objective Three 54
3.3.4 Research Design for Objective Four 54
3.4 Target Population and Sampling Strategy 55
3.5 Data Collection Instruments for Variable Validation 56
3.6 Validity and Reliability of the Data Collection Instruments 56
3.6.1 Pilot Test 56
3.6.2 Tool Validity 58
3.7 Experimental Materials 58
3.8 Data Analysis 59
3.9 Ethical Considerations 60
3.10 Chapter Summary 60
CHAPTER FOUR: RANKING AND VALIDATION OF VARIABLES DETERMINING FOG LAYER EFFICIENCY 61
4.1 Introduction 61
4.2 Ranking of Variables 61
4.2.1Experimental Design 61
4.2.2 The Experimentation 62
4.3 Ranking Output 64
4.4 Ranking Validation by Industry Experts 65
4.4.1 Data collection 66
4.4.2 Feedback from Respondents 66
4.5 Analysis of Ranking Validation 69
4.5.1 Kendall's Coefficient of Concordance 69

4.5.2 Significance of the Variables Ranking 72
4.5.3 Correlation Analysis 72
4.6 Discussion 73
4.7 Chapter Summary 75
CHAPTER FIVE: DESIGNING A DBSCAN AND KNN BASED HYBRID MODEL 77
5.1 Introduction 77
5.2 The DBSCAN Algorithm 77
5.3 The KNN Algorithm 79
5.4 The Hybrid Algorithm 81
5.4.1 Pseudocode for Hybrid the Algorithm 81
5.4.2 Flow Chart for the Hybrid Model 82
5.5 Mathematical Expression of the Hybrid Algorithm 83
5.6 Sequence Diagram for the Hybrid Model 84
5.7 Discussion 85
5.8 Chapter Summary 85
CHAPTER SIX: DEVELOPMENT OF HYBRID MODEL TO IMPROVE FOG EFFICIENCY 88
6.1 Introduction 88
6.2 Development Process of the Hybrid Model 88
6.2.1 Design Science Approach 88
6.2.2 Sequence Diagram for the Model Development 90
6.3 DBSCAN Performance 91
6.4 Experimental Evaluation of KNN 93
6.5 Experiment on the Hybrid Model 94
6.6 Comparison of the Model with Baseline Performance. 100
6.6.1 Comparison of the Model with the Baseline KNN on All Variables. 100
6.6.2 Comparison of the Model Testing Results with Baselines in Literature 103
6.6.3. Error Rate Analysis of Benchmark Values. 104
6.7 Comparison of Hybrid Model with other Existing Models 105
6.8 Discussion 106
6.9 Chapter Summary 106
CHAPTER SEVEN: EVALUATION OF THE HYBRID MODEL USING YAFS SIMULATOR 109
7.1 Introduction 109
7.2 Experimentation Process. 109
7.2.1 Setting up the YAFS Environment 109
7.2.2 Defining the Simulation Topology 110

7.2.3 Generating IoT Workloads 111
7.2.4 Implementing the Hybrid DBSCAN-KNN Model..... 111
7.2.5 Deploying the Model in YAFS 112
7.2.6 Running the Simulation 112
7.2.7 YAFS simulation Workflow 112
7.2.8 Analysis of Efficiency Metrics..... 113
7.3 Experiment Results for Simulating DBSCAN Algorithm in YAFS..... 114
7.4 Simulating KNN in YAFS..... 115
7.5 Simulating Ensemble Model in YAFS..... 117
7.6 Simulating Hybrid Model in YAFS 119
7.6.1 Simulating Packet Loss Rate on the Hybrid Model 121
7.6.2 Simulating Queue Time on the Hybrid Model..... 123
7.6.3 Simulating latency on the hybrid model..... 125
7.6.4 Simulating Channel utilization on the Hybrid model..... 127
7.6.5 Simulating Throughput on the Hybrid Model 129
7.6.6 Impact of Moderating Variables to Simulation Results 131
7.7 Results for Simulating Control Setup - KNN+DBSCAN 132
7.8 Comparison of Simulation results with baselines..... 133
7.9. Statistical Analysis..... 134
7.9.1 Hypothesis Testing (Paired t-test)..... 135
7.9.2 Regression Analysis 136
7.9.3 Confidence Intervals..... 137
7.9.4 ANOVA for Metric Interactions 138
7.10 Discussion..... 138
7.11 Chapter Summary..... 142
CHAPTER EIGHT: CONCLUSION, RECOMMENDATIONS AND FUTURE WORK..... 143
8.1 Conclusion..... 143
8.2 Recommendation 147
8.2.1 Recommendation to Practice 147
8.2.2 Recommendation to Policy..... 148
8.3 Future Work..... 149
REFERENCES 150
APPENDICES..... 168

LIST OF TABLES

Table 2.1: Clustering Algorithms 28
Table 2.4: Variables Determining Fog Efficiency..... 37
Table 3.1 Responses from pilot testing. 57
Table 4.1: Output of ranked variables from the most critical to the least critical 64
Table 4.2: Responses from industry expert ranking 70
Table 6.1: Experimental output of KNN on All Variables 94
Table 6.2: Results of Hybrid Model on all variables..... 96
Table 6.3: Results of Hybrid Model on packet loss rate. 97
Table 6.4: Results of Hybrid Model on Queue time variable. 97
Table 6.5: Results of Hybrid Model on Latency Variable 98
Table 6.6: Results of Hybrid Model on Chanel utilization variable. 99
Table 6.7: Results of Hybrid Model on Throughput variable. 99
Table 6.8: Results of Hybrid Model with the baseline KNN. 100
Table 7.1: Simulation of DBSCAN Algorithm in YAFS 114
Table 7.2: Simulation of KNN algorithm in YAFS..... 116
Table7.4 : Simulation of Ensemble KNN in YAFS 119
Table 7.5: Results from simulating DBSCAN+KNN hybrid model in YAFS 120
Table 7.6: Comparison between hybrid model, KNN and ensemble KNN baselines 134
Table 7.7: Test Results for the Model Simulation. 135

LIST OF FIGURES

Figure 2.1: IoT fog architecture..... 10
Figure 2.2: Clustering in DBSCAN..... 23
Figure 2.3: Conceptual framework..... 47
Figure 3.1 Methodological workflow for the hybrid model. 52
Figure 5.1: Flow chart for the hybrid Algorithm..... 83
Figure 5.2: Sequence diagram for the Hybrid operation..... 85
Figure 6.1: Design Science Process..... 89
Figure 6.2: Stages of the model development..... 90
Figure 6.3: DBSCAN clustering..... 92
Figure 6.4: Confusion matrix results for KNN..... 93
Figure 6.5: Comparison of accuracy for hybrid model with the baseline KNN. 101
Figure 6.6: Comparison of the model with baseline algorithms. 102
Figure 7.1: Installation of dependencies 110
Figure 7.2: Set up of topology..... 110
Figure 7.3: Setup IOT Environment 111
Figure 7.4: Data clustering..... 111
Figure 7.5: Training KNN on clustered data..... 111
Figure 7.6: Data classification..... 111
Figure 7.7: Logging of simulation platform..... 112
Figure 7.8: YAFS Simulation Workflow..... 113
Figure 7.9: Simulation of DBSCAN in YAFS 114
Figure 7.10: Confusion matrix from KNN simulation on YAFS..... 117
Figure 7.11: Confusion Matrix for Simulating Ensemble KNN 118
Figure 7.12: Clustering results on testing DBSCAN+KNN model..... 120
Figure 7.13: Confusion matrix of packet loss rate for the hybrid model..... 120
Figure 7.14: Scatter plot of packet loss rate for the hybrid model 1202
Figure 7.15: Confusion matrix of queue time for the hybrid model.....1243
Figure 7.16: Scatter plot of queue time for the hybrid model. 1244
Figure 7.17: Confusion matrix of latency for the hybrid model. 1265
Figure 7.18: Scatter plot of latency for the hybrid model. 1266
Figure 7.19: Confusion matrix of channel utilization for the hybrid model. 1287
Figure 7.20: Scatter plot of channel utilization for the hybrid model.....12828
Figure 7.21: Confusion matrix of throughput for the hybrid model.....13030
Figure 7.22: Scatter Plot of throughput for the hybrid model. 13030
Figure 7.23: Results for the control experiment KNN+DBSCAN. 13232

ACRONYMS AND ABBREVIATIONS

AMQP	Advanced Message Queuing Protocol
API	Application Interface
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
FCM	Fog Computing Model
GA	Genetic Algorithm
GB	Giga Byte
HFLO	Hybrid Fog Cloud Offloading
HTTP	Hypertext Transfer Protocol
ICT	Information Communication Technology
IEEE	Institute of Electrical and Electronics Engineers
IOT	Internet of Things
KNN	K-Nearest Neighbour
MB	Mega byte
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
NCI	Node Computing Index
NNR	nearest Neighbour Rule
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
PSO	Particle Swam Optimization
QOS	Quality of Service
REST	Representational state transfer
RF-RFE	Radom Forest Recursive Feature Elimination
RTT	Run Time Trip
SDN	Software Define Network
SLA	Service Level Agreement
SLO	Service Level Objective
SSA	Scalp Swam Algorithm
TSP	Travelling Salesman Problem
TTFB	Time to First Byte
VM	Virtual Machine
WIFI	wireless technology
YAFS	Yet Another Fog Simulator

CHAPTER ONE
INTRODUCTION

1.1 Background of the study

The drive behind exploring ways to boost efficiency in the fog layer of the Internet of Things (IoT) comes from the need to overcome challenges in traditional cloud systems like delays, limited bandwidth, and high energy use. With IoT devices producing huge amounts of real-time data, fog computing processes this data closer to where it is generated. Making this layer more efficient helps improve response times, cut down data loads, and save energy especially in areas like healthcare, smart cities, and industry. This research plays a vital role in creating smarter, faster, and more sustainable IoT systems that can keep up with today's digital demands.

The fog layer plays a key role in the IoT ecosystem by decentralizing infrastructure, placing data, processing, storage, and applications between the clouds and where the data is actually generated [1]. It essentially brings the benefits of cloud computing closer to the source and user of the data. Since both the fog and edge layers aim to move intelligence and computation nearer to the data origin, they are often used interchangeably [2]. While this shift can also support security and compliance, it primarily boosts overall productivity.

The term "fog" is inspired by the weather concept of a cloud near the ground symbolizing how fog computing brings cloud-like capabilities closer to the edge of the network [3]. The name was coined by Ginny Nichols, a product line manager at Cisco, and is now a registered term. Fog computing enables quick, real-time analytics near data sources, while more complex, long-term processing remains in the cloud. It complements rather than replaces cloud computing. Although edge devices and sensors generate and collect data, they often lack the processing and storage power needed for deeper analytics [4]. On the other hand, cloud servers can handle such tasks, but their distance may lead to delays in processing and response.

The location of intelligence and processing power is the primary distinction between fog and edge computing, according to the Open Fog Consortium [1]. Data is sent from

endpoints to a fog gateway in a strictly foggy environment, where it is processed and then sent back to the sources for transmission [2]. The endpoint or gateway in edge computing may be powerful and intelligent. Literature highlights how each device functions independently and chooses whether data to send to the cloud or a gateway for additional processing, removing points of failure. Its scalability provides a better overall view of the network than edge computing since it gathers data from several data points [1].

Fog computing's main advantage is that it can accommodate applications that need low latency, mobility, and location awareness. This capability is made achievable by the widely dispersed deployment of Fog computing systems near the end consumers [3]. Fog computing nodes must consequently have sufficient processing and storage power to meet resource-intensive user needs.

Traditionally, cloud computing has supported Internet of Things (IoT) applications by offering data storage and processing power to help businesses operate more efficiently. However, as IoT usage evolves, it’s pushing the limits of conventional cloud systems. The massive flow of data from IoT devices can overwhelm the cloud, leading to delays and inefficiencies. Sending large volumes of data at high speed to the cloud is not only expensive but also impractical [8].

As a result, processing needs to happen closer to where the data is generated. Fog computing addresses this challenge effectively by bridging the gap between IoT sensors and the cloud. It enables faster, more localized processing, especially in scenarios where cloud solutions fall short. This layered approach spanning the cloud, fog, and IoT devices offers users greater flexibility and control over their data and resources [9].

The potential uses for fog computing are numerous. One use of fog computing that is becoming more and more common is traffic control. Sensors such as traffic detection sensors have connectivity to cellular networks, cities oftenly locate computing resources close to the cell mast. These capabilities allow traffic data to be examined in real-time, allowing traffic signals to respond to shifting circumstances in real-time [4].

An example of fog computing application is in autonomous vehicles. These cars act as edge devices because they're equipped with powerful onboard computers. They must collect data from various sensors and process it instantly to make split second decisions [11]. In some cities, there's growing interest in leveraging the same processing infrastructure used in traffic lights to support autonomous vehicles. These vehicles could serve as edge devices, feeding real-time traffic data into a shared system. This information, combined with other sources, would allow the system's central computer to optimize traffic signal operations more effectively [12].

Fog computing can be thought of as a virtualized layer that sits between cloud data centers and end-user devices. It brings computing, storage, and networking closer to where data is created. Unlike traditional cloud systems, fog computing relies on a wide range of diverse, decentralized, and widely distributed devices that work together and interact directly with the network processing and storing data independently, without needing to rely on third-party servers [13].

The efficiency of the fog layer is very important in today's Internet of Things (IoT) environment because it directly affects how well connected devices and applications perform [7]. In simple terms, the fog layer acts as a bridge between the edge, where devices like sensors and smart machines collect data, and the cloud, where large-scale processing and storage take place [4]. When this middle layer is efficient, it reduces delays by processing data closer to where it is generated [19]. For example, in applications such as self-driving cars, smart healthcare, and industrial automation, even a small delay can lead to serious risks or losses.

An efficient fog layer ensures low latency, high throughput, and better management of resources like bandwidth, making real-time decision-making possible [6]. It also minimizes congestion by handling tasks locally instead of sending everything to the cloud, which reduces costs and improves reliability [11]. Moreover, efficiency in the fog layer supports scalability as the number of IoT devices continues to grow, preventing systems from being overwhelmed [14]. In essence, a well optimized fog layer creates a balance of speed, reliability, and cost effectiveness, making IoT systems more practical, safe, and beneficial for industries and everyday users alike

1.2 Problem Statement

Fog computing is essential for scaling the Internet of Things (IoT) because it moves computation closer to devices, reducing the load on distant clouds and improving responsiveness. Yet many real-world fog deployments still face performance bottlenecks that reduce reliability for latency-sensitive and mission-critical IoT applications. A central reason is that most studies optimize one or two QoS metrics in isolation rather than modelling the combined, interacting impact of the core fog-efficiency variables: packet loss rate, queue time, latency, channel utilization, and throughput. This narrow focus leaves important tradeoffs unexplored and prevents practitioners from tuning fog systems holistically.

Several works come close to this problem but stop short of a complete solution. [5] Proposes a hybrid load-balancing scheme that reduces packet loss and evens server load in fog environments, but their evaluation emphasizes packet loss and response time without modelling queue dynamics or channel utilization under heterogeneous traffic mixes. [6] presents an IIoT cloud–fog hybrid framework that lowers processing latency in industrial scenarios; however, the study assumes idealized link conditions and does not jointly analyze packet loss, throughput, or queueing behavior when traffic bursts occur. [7] Comprehensive survey catalogues resource-allocation strategies across the fog–cloud continuum and highlights inconsistent metric definitions across studies, showing that few contributions evaluate packet loss, queue time, channel utilization and throughput together.

[8] and related state of the art reviews discuss bandwidth and resource provisioning for better user experience but still leave open the integration of fine-grained link level variables like packet loss, channel contention, with node level queue models and throughput evaluation. Recent latency optimization studies advance dynamic allocation techniques across hybrid cloud boundaries, yet many validate only latency and average throughput in isolation and do not measure how improvements affect packet loss or queue time under mobility and burst IoT traffic.

Collectively, these gaps show there is no tested hybrid modelling approach that explicitly incorporates packet loss rate, queue time, latency, channel utilization, and

throughput as coupled variables, and also quantifies their trade-offs across realistic IoT traffic, mobility, and resource heterogeneity. This study develops and evaluates a hybrid model that integrates these efficiency determinants, enabling explainable trade-off analysis to improve fog layer performance for real world IoT deployments.

1.3 Objectives

This research has the following objectives.

1.3.1 Main Objective

To Implement a hybrid Model for improving efficiency in fog layer of the Internet of Things using Density-based spatial clustering of applications with noise and K-Nearest Neighbour algorithms.

1.3.2 Specific Objectives

i. To analyze the variables influencing fog layer efficiency, rank them through RF-RFE, and validate with industry experts.
ii. To design a DBSCAN and KNN based algorithm for improved fog layer efficiency.
iii. To develop a hybrid model based on the two algorithms and test its effectiveness in improving fog layer efficiency.
iv. To evaluate the hybrid model using YAFS (Yet Another Fog Simulator) and compare results with benchmarks.

1.4 Research Questions

i. Which are the fog efficiency determining variables in the existing approaches and what is their hierarchy?
ii. How to design a DBSCAN and KNN hybrid model for improved fog layer efficiency?
iii. Which fog layer efficiency metrics demonstrate improvement as a result of the development of the hybrid DBSCAN–KNN model?
iv. How does the hybrid model perform in YAFS compared to benchmarks?

1.5 Scope of the Study

This research focused on developing a hybrid model aimed at improving the efficiency of the fog layer in the Internet of Things (IoT) by addressing key performance variables: packet loss rate, queue time, latency, channel utilization, and throughput. The fog layer, which lies between IoT devices and the cloud, is critical for processing data closer to its source. However, as the number of connected devices grows, the fog layer often struggles with performance issues that can slow down or disrupt communication.

Packet loss rate is central to the study because data loss during transmission can affect the accuracy and reliability of IoT applications. The research sought to design a model that minimizes packet loss, ensuring that more complete and accurate data reaches its destination. Queue time, is about how long data packets wait in the processing queue of fog nodes. High queue times lead to delays in decision making, which can be harmful in time sensitive applications like healthcare or traffic control. The hybrid model was designed to reduce queue times by distributing workloads more intelligently.

The other focus was latency, which is the delay between sending and receiving data. Since many IoT applications depend on real time responses, the model aimed to cut down latency by processing data more efficiently at the fog layer. Channel utilization measures how effectively the available communication channels are used. Poor utilization leads to congestion and wasted bandwidth. The research explored how the hybrid model can balance traffic loads to improve channel efficiency.

Throughput was examined as a measure of how much useful data is successfully delivered over time. By improving the above variables, the model aimed to maximize throughput, supporting smoother and more reliable IoT operations.

1.6 Significance of the Study

High dimension data streams are created at the edge layer on a high scale. The large scale, high dimensionality, and evolving nature of these data streams present significant challenges for the fog layer in the edge-fog-cloud IOT continuum. They

result in reduction of efficiency especially in time critical tasks. The current models and algorithms have not addressed fog efficiency adequately. Moreover, most existing approaches are static, which cannot be tailored to the complex dynamical features of the real fog layer. Therefore, techniques that are more effective and efficient at processing continuously changing data streams are paramount to the fog computing layer. This study has come up with a model to address such delays in the fog layer thus improving its performance.

Traditionally, clustering and classification are independent approaches. The proposed model integrates the two where DBSCAN is used for preprocessing data by clustering similar data points together while identifying noise and outliers. Features extracted from each cluster is used as inputs for classification tasks. KNN classifies new data points based on their proximity to the centroids or representatives of the clusters. This increases the speed and accuracy of classification tasks thus improving the overall fog efficiency.

1.7 Contributions the Thesis

This study has made the following contributions to theory and practice:

1.7.1 Contribution to Theory

This study has made three theoretical contributions:

i. The hybrid model advances networking theory by improving fog computing in IoT through combining DBSCAN for clustering and KNN for classification. This integration enhances fog-layer performance by reducing delays, data loss, and congestion. Additionally, RF-RFE is used to rank key efficiency factors, enabling faster and better decision-making in dynamic network environments.

ii. The hybrid model strengthens decentralized data processing by improving real-time handling of data at the fog layer. Using DBSCAN for clustering and KNN for classification, it manages noisy data locally, reducing cloud dependence, delays, and network load. This enables faster responses, better scalability, and shows how machine learning enhances efficiency in complex, time-sensitive IoT environments.

iii. The model adds to distributed computing theory by showing how DBSCAN and KNN can work together to help fog nodes handle tasks more efficiently. It improves how work is shared across the system by removing irrelevant data and accurately sorting tasks based on local information, which boosts overall performance.

1.7.2 Contribution to practice

This study has made one contribution to practice:

This hybrid model processes data locally from edge devices, allowing faster decision-making without constantly depending on the cloud. This is especially important for real-time applications where quick response times are critical. By handling data at the fog layer, the model greatly improves performance, making the system more efficient and responsive.

1.8 Organization of the Thesis

The study focuses on hybrid approaches, discussing their contributions and gaps. Simulation environments are explored, emphasizing the role of synthetic datasets in testing models. Various approaches used to improve fog efficiency are reviewed. In chapter two, a systematic literature review is carried out to identify the variables which determine fog computing efficiency. Chapter three outlines the research methodology, covering research philosophy, design, target population, sampling strategy, and data collection instruments. A pilot study validates research instruments, while data analysis methods are outlined, linking each objective to the design science steps. Ethical considerations for the study are also outlined.

Chapter four ranks fog efficiency variables using RF-RFE algorithm, followed by validation by industry experts through a survey. Chapter five is about development of an algorithm from two base algorithms, DBSCANN and KNN, which is the basis for designing the hybrid model. Chapter six focuses on developing the hybrid model from the derived algorithm and experiments to test it. Chapter seven is on simulation of the model on YAFS platform. The results are analyzed to establish the significance of the entire study. Chapter eight presents the conclusion, recommendations and future work, summarizing the findings and suggesting areas of further research.

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

Fog computing is essential in the quickly developing Internet of Things space because it enables real-time data processing at the network's edge, reducing latency and enhancing application efficiency [9] . Key efficiency indicators have a significant impact on the fog layer's effectiveness. To make sure fog computing systems can effectively handle and analyse data from a variety of IoT devices, it is essential to recognize and modify these features [10]. In addition to enhancing the functionality of specific IoT applications, an understanding of these factors increases the overall reliability and scalability of fog computing systems in a data-intensive setting.

Fog layer extends the concepts of cloud computing to the network edge by distributing processing resources over numerous nodes, including routers, switches, and local servers [11]. By processing data closer to its source, this distribution aims to improve data efficiency and decrease reaction times [12]. Even with these developments, latency issues still exist, which affects fog computing systems' effectiveness and performance. Optimizing these systems requires an understanding of the different variables and how they affect fog performance.

2.2 The IOT Architecture

Cisco first introduced fog computing as a way to expand the cloud to the network's edge [13]. The routers, gateways, access points, and switches that comprise the fog nodes are able to manage data by offering networking, computing, and storage as a service closer to the edge of the network. The network has significantly lower latency as compared to the cloud [14]. Since data can be pre-processed on the Fog layer, not all data needs to be moved to the cloud. As a result, less energy, delay, and bandwidth are required. The distributed method is made easier to handle by fog, which makes equipment more mobile [15].

The heterogeneous nature of fog devices makes it challenging to develop an architecture and create plans for resource distribution and task scheduling [36]. Since

fog devices are local to the network and don't require an internet connection, they can be used in the event of internet disruptions. With the introduction of Fog, security standards are implemented directly on the Fog servers, protecting data and preserving privacy without the help of a third party. IoT devices can receive services from fog and the edge using very similar methods [16]. On the other hand, the edge is generally considered to refer to a technique that functions at the edge of the network.

The goal is to complete the computation as close to the end devices as feasible, usually with only one hop, but sometimes with more. The edge may pre-process, filter, and aggregate data since it is positioned between the cloud and the IOT. While fog focuses primarily on infrastructure, edge computing is object-oriented [17]. The outstanding difference is where the data computation is done. Figure 2.1 shows the IOT architecture as authored by [34]

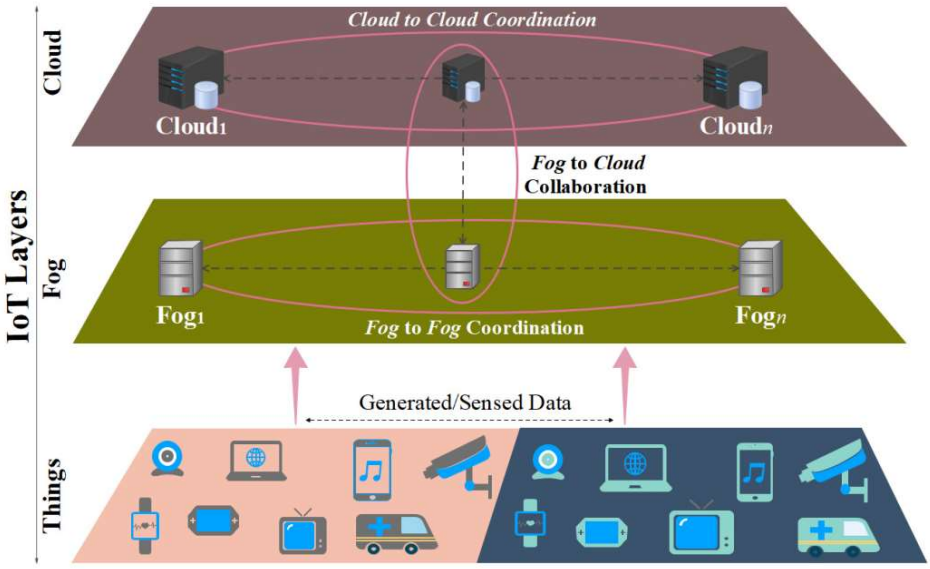


Figure 2.1: IoT fog architecture

2.2.1 Thing Layer

This layer, which is also called the perception layer, in tune with both the physical sensors' and users' environments. The networked devices are housed in this layer, which also permits data sharing and sensing. It includes the linked IoT sensors, such as environmental, blood-oxygen, and heart-rate sensors, among others. The primary function of these widely dispersed sensors and devices in this layer is to collect data

from adjacent real-world objects or events and transmit towards the top layer for processing. There is a communication protocol for every gadget or sensor. For the objects to communicate data processing requests to higher levels using the generated/sensed data, this communication protocol is required. For the objects to communicate data processing requests to higher levels using the generated/sensed data, this communication protocol is required.

2.2.2 Fog Layer

This is situated at the layer boundary of the item. In this layer, it is best to place multiple distributed fog nodes [18] in close proximity to data sources, or objects. Any device that meets these specifications is referred to as a fog node: The communication protocol; the processing and storage capability; and the distance to the sources. Access points, switchers, routers, base stations, gateways, and the designed fog nodes are examples of this [19]. The data at the edge layer is processed here.

In order to expedite and simplify the process of gathering and processing data, fog nodes are positioned at the edge of networks [20]. Each fog node is therefore equipped with internal processing resources, data storage, and network communication capabilities in order to connect items and the cloud within the IoT network [21]. In order to conveniently and independently gather and evaluate data, communication between the fog nodes with other objects happens [22]. For boosting processing power for enhanced services, operations and storage capacity, fog nodes are also in charge of communicating and cooperating with the cloud.

2.2.3 Cloud Layer

The layer offers centralized network control and monitoring. It provides constant, appropriate, and helpful network access to shared resources. It is composed of multiple high speed servers [23]. This layer offers robust computation and storage capabilities to support long term data storage and comprehensive computer analysis. Processing massive amounts of data and sophisticated machine learning tasks like analysis and training are best done in cloud data centers.

Bandwidth, Latency, privacy, security and, and context awareness are few of the limitations brought about by the distance from the cloud servers and the edge devices.

Real-time applications such as IIoT, e-health, and others require low latency, even a few milliseconds. A delay of 250 s to 10 Ms and 50 Ms is advised for smart cars and autonomous factories [24]. There is communication latency when Internet of Things devices transfer duties to the cloud. Scholars predict that going forward, the amount of data emanating from IoT devices will increase [23].

One gigabyte of data will be produced by self-driving cars every second, and over a petabyte of data will be produced annually by the US smart grid by the year 2025 [25]. Large amounts of bandwidth will be needed for data sending and receiving to and from the cloud in order to evaluate all of the data stored there. Energy from the nodes is essential for monitoring and responding to the environment. Servers will need a lot of energy to cool and analyse data that is offloaded to the cloud [26].

To use cloud services, we need to be connected to the internet, since the cloud relies on that connection to function. However, technical problems or poor internet access can sometimes block users from reaching those services, making them unavailable [38]. Placing fog computing at the edge of the network helps solve this by improving device mobility, reducing delays, and giving better awareness of where IoT devices are located. Fog nodes can handle data in two ways: they can either process it themselves or filter it first before sending it on to the cloud for further processing [39].

2.3 Theoretical Foundations

Fog computing was created to solve the problems with latency, bandwidth, and real-time data processing that cloud computing presents [27]. Three fundamental theories; edge computing theory, distributed computing theory, and network theory form the basis of fog computing [18]. Together, these theories allow fog computing to offer low-latency, scalable, and effective solutions to meet the growing demands of data-intensive applications and Internet of Things (IoT) devices.

Edge Computing Theory, emphasizes the necessity of bringing computation closer to the data source. Despite its strength, traditional cloud computing has latency and bandwidth problems when processing data that must be sent over large distances [27]. This is fixed by edge computing, which speeds up reaction times by processing data

locally on devices or close to the data source. By adding intermediary fog nodes that supply extra computational resources between the edge and the cloud, fog computing expands on this idea [28]. For applications like industrial IoT, healthcare monitoring systems, and driverless cars, these nodes' ability to process data in real-time eases the strain on cloud servers and guarantees quicker decision-making [29].

Decentralized data processing is made possible by distributed computing theory, which is important to fog computing architecture. Fog computing enables parallel processing, fault tolerance, and scalability by distributing workloads and resources among several nodes. For handling complicated workloads that call for the cooperation of multiple devices or nodes inside a network, distributed computing theory is key [30]. Fog systems can operate efficiently due to optimal load balancing, which distributes work among fog nodes and dynamically allocates processing based on available resources [31]. Because work may be redirected in the event of node failures, distributed computing also enables improved fault tolerance, increasing system stability and reducing downtime [32].

Network theory is vital in fog computing because it governs how data moves and how communication happens between edge devices, fog nodes, and the cloud. Since IoT devices generate huge amounts of data, fog computing depends on a strong and efficient network structure to ensure this data reaches the right processing nodes. This involves managing bandwidth effectively, reducing delays, and optimizing data channels [33]. To achieve this, fog systems use strategies such as dynamic path adjustments and hierarchical routing, which help design network topologies and routing protocols that make data processing more efficient [34]. Network theory also emphasizes the importance of fault tolerance, ensuring that communication networks remain reliable even when problems occur.

These three theories work together to provide fog computing's theoretical underpinnings, allowing it to meet the increasing need for distributed systems to handle data in real-time, scalable, and effective ways. By processing data at the edge and intermediate fog nodes, fog computing improves system efficiency and reliability by lowering dependency on centralized cloud infrastructures. Integrating these

theories will be essential to creating computer systems that are more robust and efficient as IoT applications continue to expand.

2.4 Efficiency of the Fog Layer

The exponential growth of Internet of Things (IoT) devices has led to a massive surge in data generation, with over 2.5 quintillion bytes created daily and projections estimating 1.7MB per person per second by 2025 [51]. Cloud computing, although initially sufficient for processing and storing this data, is increasingly strained resulting in higher latency and reduced responsiveness [23], [52]. Fog computing addresses this by bringing computational resources closer to the data source, reducing bandwidth usage, improving response time, and enabling real-time processing [53], [54]. Unlike cloud infrastructures, fog systems face unique challenges including resource heterogeneity, power limitations, and the absence of standardized benchmarks for evaluating efficiency strategies.

Despite the existence of hybrid models aimed at optimizing fog environments, most prioritize objectives like energy consumption, cost minimization, or latency reduction [56], [57], [58]. Models such as dynamic provisioning for gaming [59] or energy-aware solutions for latency-sensitive applications [60] often focus on individual stakeholder needs, without a holistic view of system level fog efficiency. Service Level Objectives (SLOs) derived from Quality of Service (QoS) metrics are crucial for evaluating such models within formal Service Level Agreements (SLAs). However, the complexity of differing goals, conflicting metrics, and multi-tiered infrastructures complicates consistent performance evaluation.

Efficiency evaluation is therefore essential for determining whether improvement strategies fulfill intended outcomes [61], [62]. Existing hybrid models fall short in systematically measuring fog efficiency using comprehensive, domain specific metrics. This gap underscores the importance of developing and evaluating robust, context aware models tailored to fog computing’s unique operational and architectural demands.

The dynamic qualities of the data generated at the edge layer, such as volume, velocity, variety, variance, and veracity, make the existing data analysis techniques useless. Real-time dynamic data streams from the sensor layer are provided by the fog layer. This stream data is made up of a data sequence that is updated continuously and is arranged chronologically [35]. The present methods for handling such data on storage center on cloud servers; however, because of their remote location from the end devices, these servers have a number of drawbacks, including bandwidth, latency, context awareness and privacy [24].

Despite its potential, fog computing has run into problems that have caused major efficiency deterioration and service interruptions [36]. Enhancing efficiency through intelligent solution for time-sensitive and dependable flows in hybrid fog computing IOT systems [37]. To increase network observability, the suggested method included a unique technique for deploying SDN switches in between conventional switches. Additionally, it featured a K-nearest neighbor regression approach that used historical data to forecast the dependability of aged links at the link controllers in real time.

This enhances system efficiency by empowering the controller for quick decision making. Additionally, it featured the time-sensitive and dependable deep deterministic policy gradient algorithm (RTDDPG), which determines the best forwarding paths in hybrid Software Defined Network (SDN) for traffic flows that are crucial to time and are produced by Internet of Things applications. Several data transfer metrics were reviewed. Latency, tthroughput, response time, channel utilization, packet loss rate idle time, speed, queue time, and necessary time were among them. The strategy achieved a total packet loss rate of 15.5%, an aggregate response time of 151.379 MS, and an aggregate system latency of 140.05 sec. The average total channel use was 96.221% and the average speed was 237.06 KB/sec for files with sizes between 256 KB and 16 MB.

2.5 Data Processing in the Fog Computing

The purpose of categorization in data mining is to allocate items within aggregation to particular groups [38]. The objective is to accurately predict the target class for each case in the data. In the data mining process, big data sets are first sorted in order to

identify patterns and make connections that allow for data analysis and problem solving. Classification identifies which of a collection of categories a new measurement belongs to, based on a training set of data that comprising observations whose category membership is known.

To predict class during the classification process, a classifier is required. There are two steps involved, like: The purpose of classification in data mining is to allocate items within a grouping to particular groups [39]. The target class for each case in the data is to accurately predict. In this process, big data sets are first sorted in order to identify patterns and make connections that allow for data analysis and problem solving.

Machine learning classifiers exist in a variety of forms. One example is the Decision Tree, which models decisions and predicts their possible outcomes using a tree-like structure. The idea that a (natural) class's purpose is to predict the values of its members' features forms the basis of Bayesian classifiers [40]. Neural networks are built on artificial neurons, which are collections of connected units or nodes that resemble neurons in a human brain. In contrast, the K-NN classifier uses closeness to categorize the grouping each data point.

It is a non-parametric classifier of the supervised learning type [41]. Furthermore, supervised learning models like Support Vector Machines examine data for regression and classification using associated learning methods. Linear regression can depend on the value of another variable, to forecast the dependent variable. The variable used is known as the independent variable.

Fog computing based scheduling of the various tasks of IOT, an applications quantumized load strategy (QLS_IoTA) was proposed in [42]. To evaluate the approximate processing capacity the authors developed a Node Computing Index (NCI), which is static. In order to manage the diverse real-time job and anticipate the optimal fog node, the authors also built the QCI model, a neural network-based methodology. In every fog node, they employed 5, 10, 15, and 20 distinct scenarios to replicate a range of tasks from the Google Job databases. A hybrid Fog to Cloud

(F2C) study was presented in [43] as a means of concurrently placing and executing services.

When examining dispersed service locations, the authors' model considers both sequential and parallel service delivery mechanisms. While band width consumption showed negative variance, reduced service response times resulted in a considerable efficiency in service delivery. In [44], the authors presented a number of criteria to be used in conjunction with the primary task offloading paradigm for fog computing. To manage job allocation certainty in a fog cloud computing environment, a fuzzy clustering was employed.

To solve the issue of task allocation uncertainty, bipartite graphs were used. The cloud servers' processing power was raised which consequently raised the tasks' overall cost. This solution speeds up the completion of the work since it uses higher DAG levels and fewer fog nodes. Authors in [10] introduced an integer programming model that treated scheduling as an optimization problem in order to lower the gross service request delay of IoT service requests (SIoTA_MLHF).

A modified version of the genetic algorithm is employed as a formula to plan IoT queries and reduce latency overall. The improved GA utilizes the environment's dynamic component to test the simulation's results. In [68], a novel concept for enhancing the information-centric cloud-fog computing system (CFC_ICIoT) IoT application dispatching functionalities is proposed. The dispatcher is designed using two unique features: resource scheduling and task classification. The CFC_ICIoT solution achieves superior balancing by catering to the QoS requirements of individual applications.

Thus, the sight made during the previously described advancement of relevant study activity in the designated subject led to the recognition of the enormous optimization for job scheduling. According to [45], if the tasks are grouped based on their quality of service (QoS) needs, it becomes easy to find and assign a suitable cloud or fog node for executing the task. This results in a decrease in processing and overall make span time.

The closest classification research was carried out by [46], who created a Context Aware Information Classification in Fog Computing. This notion was present in the fogging. However, instead of taking into account the massive amount of sensory input into fog for computation, just the appropriate information was required for calculating and subsequent processing by the inclusion of context sensitive information classifier [47]. By considering K-NN& ID3 classifiers and establishing effective connectivity between sensors and cloud servers, the proposed model enhances the functionality of fog computing and reduces latency issues.

In this case, a scenario where the fog nodes are already aware of the information to be accepted is described using context aware fogging. When it comes to sorting through the multitude of incoming sources of information and selecting the relevant ones for the user or system, context awareness is essential [48]. In this case, the KNN classifier was considered to classify the data. They learned that the basic principle of KNN is to build a network by aggregating the closest neighbours of a particular value, or weight.

It applies to both text and multimedia data and is most successful when the source side is giving more information. Initially, the classifier is assigned the label to gather the required information from the incoming large-scale data set. The study only considered the two categories of data: non-confidential and confidential. Furthermore, it envisaged edge layer data processing, where devices are not equipped to handle huge data chunks.

Clustering is a popular data mining technique which divides collections of abstract elements into classes of similar objects [40]. Several subsets of data can be created with the use of clustering. The data objects that comprise these clusters have low levels of intra- and high levels of inter-similarity. The partitioning method is the process of breaking up an input data set into many segments, or partitions. There are numerous divisions into which clustering techniques can be divided [49]. Each processing node in the system executes an operation on a subset of the data set as an alternative of the entire data set before moving on to the next one.

The hierarchical approach uses only the predefined inter-cluster distance. An S-point set of n points is grouped in this way. Initially, every point is considered as a separate

cluster. As long as there are two or more clusters and (C, C′) is the minimum for all cluster pairs, the C, C′ of two clusters can be consolidated into one cluster. A group or cluster in a data space is conceptualized as a continuous area with a high point density, separated from neighbouring groups by sparse areas.

Unsupervised machine learning techniques that identify unique clusters in the data are called density-based approaches. The grid-based approach divides the instance space into a grid structure. Then, rather of employing discrete data points as the fundamental units. The model-based approach is a methodical, quantitative technique to solving problems pertaining to the design of signal processing, communication systems and complex control. It is used in many motion control, automotive, aerospace, and industrial equipment applications. The constraints based method, a mathematical approach with scientific validation, limits the outcome of each decision to a minimum and maximum range of limitations (+/- infinite is allowed) [50]. Decision variables that share a constraint must also have their solution values fit inside the limitations of that constraint.

In dynamic clustering, two concepts are considered, a kind of incremental machine learning: the taught model's self-adaptation (in terms of parameters and structure) and the incrementalism involved in the learning techniques applied to build the model for clustering. Thus, memory constraints and the need for extensive retraining are addressed by instrumentality. Thus, dynamic attributes (structural elements and such behaviour) of the model to be learned are documented by modifying the existing models [51].

As observed, dynamic clustering attracts the attention of numerous academic communities in recent years, including data mining, fuzzy theory, control, pattern recognition, machine learning, etc. Dynamic clustering can be useful in a wide range of additional clustering scenarios, such as those involving extremely big data sets, incomplete data, noisy data, data streams, imbalanced data, and structured data [18]. For this reason, the challenges are very fascinating. The possibility that clustering is a single stage in a complex multi-step system makes it much more relevant.

Control, surveillance, artificial vision, and such applications rely on the clustering output to determine how effective the overall multi-step system is. When the time component is introduced, such applications become dynamic and clustering becomes challenging [50]. Dynamic clustering of heterogeneous, distributed, spatial-temporal data parallel, in ubiquitous environments is a challenge for many applications. Increasing global efficiency has been the focus of much of the prior literature when several devices collaborate to off loaf the fog [52], [53] .

[36] addressed the co-operator matching execution problem for local mobile. Although they featured four alternatives for task execution, the addition of D2D-assisted fog offloaded execution in comparison to [51], which permits transmission of tasks to the fog server via a neighbouring device. Under certain constraints, the authors of [38] concentrated on the overall energy efficiency of fog unloading. [53] Studied cooperative task offloading in fog-radio access networks to reduce service delay. The objective of [54] was to maximize network output in a layered network with fog support.

While [4] examined the energy-efficient offloading of fog computing, analysing execution time and energy consumption using the queuing theory. Nevertheless, none of these authors discussed the crucial factors governing the dynamic fog layer's performance. Much emphasis has been focused on the incremental clustering problem for the dynamic data. One example of an incremental clustering technique is using algorithms based on K-centroids [55]. Usually, they start by detecting the centers and clusters of the available data using traditional K-centroids techniques, and then they process the data piecemeal to arrive at the conclusions.

Their primary advantage is simplicity; nevertheless, they are not suitable for non-spherical, unbalanced data because they usually involve predefining the number of clustering patterns. The hierarchical incremental clustering techniques proposed in [46] are not initialization-sensitive and can handle no spherical data, but they frequently maintain a hierarchy. When an object is added to a hierarchy, a number of linked hierarchies are reorganized in the regions that it affects, which can be costly computationally. Density-based incremental clustering exemplary additive clustering

technique can overcome the challenges of discovering clusters of any shapes and managing noise [56].

They can segment newly arrived data objects sequentially, but the update mechanism needs to sift through the full data set, which is time-consuming. Another disadvantage of the approaches is that they make choosing the appropriate input settings difficult. AP clustering [57], done by transmitting messages on simultaneously, is extended to the dynamic environment in two ways: IAP based on Nearest Neighbour Assignment (IAPNA) and IAP based on K-medoids (IAPKM). However, there are several disadvantages to these methods. For example, IAPNA consumes a lot of memory, while IAPKM may not adjust the number of clusters in response to the arriving objects.

In order to analyse the vast amounts of dynamic data, [58] created the ICFS clustering based on k-medoids (ICFSKM) approach by modifying the CFS clustering method. Nevertheless, despite the stated positive results, it becomes ineffective when dealing with round shape and imbalanced data. Some other methods employ soft or fuzzy clustering including: FCM with multiple medoids, the single pass FCM [59], the online FCM [35], the kernel FCM [60], and the online nonnegative matrix factorization [61]. When handling data objects that might not be easily distinguished during the processing of dynamic data, [62].

None of these methods are suitable for managing large amounts of dynamic data because each one dictates the number of clusters to be present. Once classification is done, the supervised learning technique, KNN transfers all computations to lazy learning [14]. The foundation of the KNN technique is the idea that if the majority of samples fit into a given category and a sample has k all but comparable neighbours, then the sample itself falls into that category as well. Usually used for classification tasks, the voting strategy chooses the label that appears the most frequently in the k samples as the anticipation result.

Regression tasks can be handled using average strategy, which takes the k-sample mean of the real value output marking as the prediction result. The results of the categorization vary significantly depending on the value of k. Once different spacing computation methods are applied, the nearest neighbours could differ dramatically,

leading to significantly different classification results. The nearest neighbour rule serves as the hypothetical foundation for KNN classification [63]. Assume a hypothetical scenario where x is the named place and y is the closest point.

$$P^* \leq P \leq P^*\left(2 - \frac{c}{c-1}P^*\right)$$

Where c is the number of classes, P is the Bayes error rate, and P* is the nearest neighbour error rate. The error rate of the nearest neighbour is less than double that of the Bayes error rate when there are several points. The mistake when twice is likewise quite minimal, considering how little the error of the best probable optimal assignment is. In cases of many data points, there is a good chance that the labels for x and y are the same. Equation (i) demonstrates that even though the nearest neighbour method is quite simple, the Bayesian optimal classifier's error rate is not more than twice as high.

The primary challenge when handling massive amounts of data is dividing it up across many computer resources and using distributed parallel computing to optimize each resource's efficiency and boost processing power [64]. The search time for k-NN can be reduced if the data can be split up into smaller sections and the query can be executed in each section. In addition, it is possible to search different sets of data simultaneously. Distributed computing and storage constitute the core technologies for processing massive volumes of data [65]. The trial data are first computed on KNNB then distributively saved on each storage node when doing a KNN search on a large volume of data. The global KNN is then found by concurrently searching for the k-NN within the boundaries of KNNB.

First, data is arranged correctly on different storage nodes. Next, a number of storage nodes are put through a concurrent KNN search with KNNB on each node in order to restrict the scope of the k-NN search. As a result, huge flows of data can be processed effectively through the distributed storage and computation-based KNN approach [66]. The DBSCAN algorithm finds the dense zone by putting together data points which are close to one another based on the distance.

DBSCAN finds any kind of cluster. Using 'densely arranged' data points, it forms a cluster [10]. It can identify clusters in large spatial datasets by looking at the local

density of the data points. The most interesting feature of DBSCAN clustering is how resilient it is to outliers. Furthermore, it does not require us to know ahead of time how many clusters would be present, in contrast to K-Means, which requires us to identify the number of centroids [67].

DBSCAN uses two parameters: minPoints and epsilon. Epsilon is the radius of the circle that has to be closed around each data point in order to assess the density [68]. MinPoints is the absolute minimal number of points required at the internal circle in order for that data point to be designated as a Core point. The circle becomes an epsilon-radius hypersphere in higher dimensions [69]. The minPoints is the minimal number of data points required inside the hyperspace.

An epsilon-radius circle surrounds each data point in DBSCAN before it is split into three categories: Border point, Core point and Noise [64]. When a certain minimum number of points (called "minPoints") are found nearby a data point, that point is considered to be a core point. Any data point with fewer points than minPoints and not surrounded by other data points inside an epsilon radius is considered a border point. A cluster with a minimum of three points created using DBCAN is shown in Figure 2.2, as authored by [82]. Here, an equal-epsilon-radius circle drawn around each data point. The development of spatial clusters is aided by these two elements.

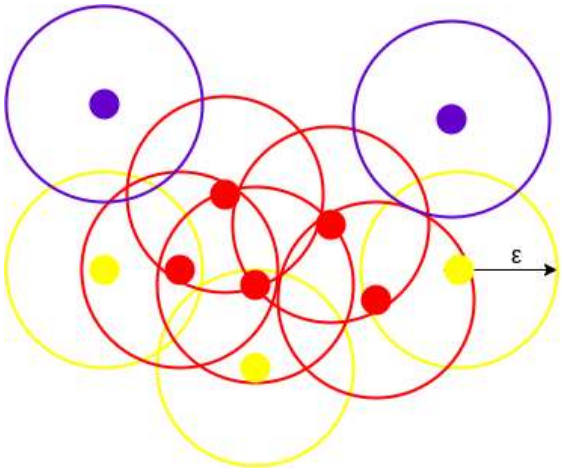


Figure 2.2: Clustering in DBSCAN

Core points are all the data points that are displayed in red and involve at least three other points in the circle, including the self [70]. The center of the circle and all data points outside the circle that are larger than zero but smaller than three are called

border points. The colour yellow is a symbol for them. Data points that are inside the circle but do not include any other points are referred to as noise, and they are represented by the colour purple [71].

DBSCAN uses Euclidean distance to find data points in space, while alternative methods (such the great circle distance for geographical data) can also be applied [72]. Furthermore, compared to other approaches that call for many full dataset scans, it just needs to do a single scan. One of DBSCAN's main benefits is its ability to identify irregularly formed clusters [73]. Of all the commonly used clustering techniques, DBSCAN is one of the algorithms that makes the fewest assumptions about the geometry of clusters. As a result, clusters with odd or irregular shapes, such as ring-shaped clusters, can be found using DBSCAN strong to extremes [74].

One important benefit of DBSCAN is its ability to detect outliers and remove them entirely from clusters. This suggests that DBSCAN, is great for datasets with a lot of outliers and is very robust to outliers and does not require the specification of the number of clusters [2]. Another advantage of DBSCAN is that it does not ask the user to specify how many clusters they want. Alternatively, DBSCAN determines the quantity of clusters quickly [75]. This is great while hard to approximate the total clusters that should exist intuitively [76].

DBSCAN is less susceptible to initial conditions, such as the seed used and the dataset's observation order, than certain other clustering methods. While some locations on cluster borders may shift as initial conditions change, most observations should remain within the same cluster [77]. [78] Developed a method called GriDBSCAN to enhance DBSCAN's efficiency by using grid merging and partitioning. This method produced great efficiency with the advantage of large parallelism. However, this approach is not suitable for high-dimensional data for the redundancy outcome of this algorithm plummets exponentially.

A technique named approximate DBSCAN, which was based on a grid model and had great complexity O (n) in low dimension for big data sets, as reported by Gan and Tao [43]. It creates to an O (n2) algorithm of a high even comparative high-dimensional data space, though. Parallel GridDBSCAN and GMDBSCAN also use grid-based

DBSCAN. However, this strategy is not applicable because it is limited to handling data with a particular density.

AnyDBC [1] reduces the data into smaller linked subsets known as primitive clusters in order for reducing propagation of the label time. It then labels objects based on connected components of these native clusters [79]. It does data indexing to expedite range query execution and runs a significant number of less range queries than DBSCAN while maintaining DBSCAN's accuracy [53]. This method can speed up query processing, but it cannot guarantee improved efficiency because it is not very good at finding the nearest neighbor in high-dimensional areas.

2.6 Hybrid Approaches in IOT

In fog computing, hybrid machine learning models and algorithms integrate various machine learning methodologies, techniques, or algorithms to improve dependability and efficiency. [80]. To solve particular problems in fog computing settings, these hybrid models combine the best features of several machine learning techniques [81]. From their work, they realised that neural networks are strong good for pattern identification and learning, whereas fuzzy logic offers a framework for managing uncertainty and imprecision in data.

[71] Focused on Genetic Algorithms with Reinforcement Learning. In their approach, wwhile reinforcement learning (RL) allows agents to acquire optimal policies through trial and error, genetic algorithms (GAs) offer a metaheuristic optimization technique for exploring huge solution spaces and locating near-optimal solutions. Fog computing uses a hybrid of reinforcement learning and genetic algorithms to provide adaptive resource allocation, task scheduling, and system optimization in uncertain and dynamic situations. However, they encountered some challenges in this model.

Slow convergence rates and significant processing overhead are common drawbacks of genetic algorithms, particularly in the context of challenging optimization tasks. Also, developing suitable reward functions and genetic operators, as well as striking a balance between exploration and exploitation, become more difficult when combining genetic algorithms and reinforcement learning.

Several hybrid models have been implemented in fog computing environments to improve performance, decision-making, and adaptability under dynamic IoT conditions. One such model is the Support Vector Machines (SVMs) with Decision Trees approach developed by Liang [12]. This hybrid leverages the interpretability of decision trees and the strong classification and regression capabilities of SVMs.

In fog environments, this combination improves accuracy and decision-making by blending human-readable rules with mathematically robust boundaries [82]. However, the model faced limitations such as reduced generalization efficiency and increased complexity, particularly when handling large-scale or high-dimensional data, where standalone models could sometimes outperform the hybrid.

Another model, developed by [85], combines Bayesian Networks with Markov Models. This hybrid is useful in fog systems for predictive analytics and anomaly detection in sequential and uncertain environments. Markov models capture temporal dynamics, while Bayesian networks handle probabilistic reasoning and causality [83]. Despite their strengths, this model suffers from scalability and computational overhead due to the exponential growth of states and probabilities, making it less practical for large datasets or complex fog architectures without efficient approximation methods [53].

[12] Explored the integration of Ensemble Learning with Meta-Learning in fog computing. Ensemble methods increase robustness by combining multiple learners, while meta-learning adapts models based on past learning experiences. This hybrid allows adaptive model selection and pruning across diverse datasets and fog scenarios [84]. Nevertheless, integrating these techniques increases training complexity and computational demand. It also requires careful consideration of model diversity, performance metrics, and stability to ensure efficiency in fog applications [85]. These challenges highlight the trade-offs between performance gains and system complexity in fog-based hybrid models.

2.7 Clustering Algorithms

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm is rooted in density-based clustering theory, which identifies clusters as high-density regions separated by low-density areas in a dataset [86]. Unlike centroid-based methods, DBSCAN does not assume spherical clusters and can detect arbitrary shapes, making it robust for anomaly detection and noisy data environments [82]. Its theoretical foundation relies on two key parameters: ε (eps), defining the neighbourhood radius, and minPts, the minimum number of points to form a dense region. DBSCAN excels in unsupervised learning by distinguishing core, border, and noise points, making it ideal for pre-processing tasks in complex datasets.

On the other hand, KNN is grounded in instance-based learning and the principle of local approximation, where predictions are made based on the majority class or average of the *k* closest data points [13]. KNN assumes that similar instances exist in close proximity within the feature space, leveraging distance metrics for classification or regression. While simple, KNN's efficiency heavily depends on data quality and feature scaling, and it suffers from the curse of dimensionality in high-dimensional spaces.

The integration of DBSCAN and KNN creates a synergistic model where DBSCAN's noise reduction capability enhances KNN's classification accuracy [87]. DBSCAN first filters out outliers and defines dense clusters, allowing KNN to operate on a refined dataset with clearer decision boundaries [14]. This hybrid approach is particularly effective in fog computing, where network efficiency metrics (latency, throughput) often contain noise.

By combining DBSCAN's unsupervised clustering with KNN's supervised classification, the hybrid model improves predictive reliability while maintaining interpretability [67]. This theoretical complementation addresses limitations of standalone algorithms, offering a robust framework for dynamic, real-time decision-making in distributed systems [88].

Different clustering algorithms use different mechanisms and each have its limitations. Table2.1 provides a summary of the different clustering algorithms, their advantages and challenges.

Table 2.1: Clustering Algorithms

	Clustering Algorithm	Description	Strengths	Limitations
1	DBSCAN [89]	A density-based algorithm that groups points in high-density regions, while identifying low-density points as noise.	Does not require the number of clusters to be specified in advance. Can detect noise and outliers. Works well with irregularly shaped clusters. Can handle clusters of varying shapes and sizes.	Sensitive to the choice of parameters (minPts, ε). Struggles with clusters of varying density.
2	K-Means [15]	A centroid-based algorithm that partitions the data into k clusters by minimizing the sum of squared distances from points to their assigned cluster centers.	Simple and easy to implement. Efficient for large datasets. Works well when clusters are spherical and of similar sizes.	Requires the number of clusters (k) to be known in advance. Assumes spherical clusters. Sensitive to outliers.
3	Agglomerative Hierarchical Clustering [90]	A bottom-up approach where each point is initially a separate cluster, and clusters are merged iteratively based on distance or similarity.	Does not require the number of clusters to be specified. Can create a hierarchy of clusters, useful for multi-level clustering.	Computationally expensive, especially for large datasets. Sensitive to the distance metric used.
4	Mean Shift [91]	A centroid-based algorithm that seeks to find dense areas of data points by iteratively shifting the centroid to the mean of the points in a given window.	Does not require the number of clusters to be specified. Can handle arbitrary-shaped clusters. Works well for spatial data.	Computationally expensive for large datasets. Requires bandwidth parameter (smoothing parameter) to be set.
5	Gaussian Mixture Model (GMM) [92]	A probabilistic model that assumes data is generated from a mixture of several Gaussian distributions, using the Expectation-Maximization (EM) algorithm for estimation.	Can handle elliptical-shaped clusters. Provides probabilistic cluster assignments. Flexible in terms of cluster shape.	Requires the number of clusters to be specified. Sensitive to initialization. Assumes data follows a Gaussian distribution.
6	Spectral Clustering [68]	A graph-based clustering method that uses the eigenvalues of a similarity matrix to reduce dimensionality before clustering.	Effective for non-linearly separable data. Can capture complex relationships between points.	Computationally expensive. Requires a similarity matrix, which can be large for high-dimensional data. Requires the number of clusters to be known.

In the dynamic, decentralized world of fog computing, the ability of DBSCAN to handle noise and find clusters of different sizes and shapes makes it a perfect clustering algorithm to hybridize with KNN for fog efficiency improvement [85]. DBSCAN's strengths include its capacity to process high-dimensional data with different densities, its effectiveness in finding outliers, and its independence from the number of clusters. These characteristics make DBSCAN an excellent choice for preparing data in fog computing, where data drastically differs among edge devices [89].

By lessening the effect of noise and offering distinct clusters for more precise predictions, DBSCAN can improve the classification stage when used in conjunction with KNN [75]. By concentrating on significant data points, DBSCAN's preprocessing helps KNN, a straightforward but efficient classifier, achieve quicker and more accurate classification.

Further supporting its integration with KNN is DBSCAN's capacity to handle imbalanced data, which guarantees that the model maintains its efficacy even in the presence of skewed data distributions, which are typical in real-time fog computing scenarios [75]. Thus, this hybrid technique greatly enhances fog efficiency in IoT contexts by combining the classification efficiency of KNN with the clustering characteristics of DBSCAN.

2.8 Classification Algorithms

Recent research has examined how well the K-Nearest Neighbors (KNN) algorithm performs in fog computing settings, emphasizing its effectiveness and versatility across a range of situations [93]. Researchers assessed KNN's efficiency in intrusion detection systems (IDS) designed for fog computing in the report "Classifier Efficiency Evaluation for Lightweight IDS Using Fog Computing" [94].

To increase the classifier's effectiveness, they used feature selection strategies like Mutual Information (MI) and Principal Component Analysis (PCA). The findings showed that KNN might function as a lightweight intrusion detection system (IDS) in fog computing environments by balancing accuracy and computational resource consumption when optimized using these feature selection techniques [46].

A different study that evaluated different KNN variations across numerous datasets was titled "Comparative Efficiency Analysis of K-Nearest Neighbor (KNN) Algorithm and Its Different Variants for Disease Prediction" [95]. With an average accuracy of 83.62%, the Hassanat KNN version outperformed the ensemble method KNN, which came in second at 82.34%. Although the focus of this study was disease prediction, the knowledge gained about KNN's efficiency and its variations is useful for fog computing applications, particularly those where real-time processing and resource limitations are crucial [96].

Furthermore, the use of AI and machine learning techniques, such as KNN, in resource management within fog and edge computing environments was covered in the systematic review "AI-based Fog and Edge Computing: A Systematic Review, Taxonomy and Future Directions" [97]. The assessment pointed out issues like high variation and the requirement for online training, indicating that although KNN is a good choice, the dynamic nature of fog computing settings may have an impact on how well it performs [77].

All of these studies show that KNN may be used successfully in fog computing platforms, especially when it is optimized using suitable feature selection techniques and customized to meet the needs of the application. However, for its successful implementation, factors like computing efficiency and flexibility to changing settings are essential [67].

Table 2.2 compares the efficiency of Classic KNN and Ensemble KNN across multiple datasets (D1–D8) using three key evaluation metrics: Accuracy, Precision, and Recall [79]. The results indicate that Ensemble KNN generally outperforms Classic KNN in terms of accuracy and precision while showing slight variations in recall performance.

Table 2.2 Benchmarks of KNN in Various Datasets.

	Accuracy		Precision		Recall	
Dataset	**Classic KNN**	**Ensemble KNN**	**Classic KNN**	**Ensemble KNN**	**Classic KNN**	**Ensemble KNN**
D1	76.35	77.03	80.9	78.57	80	85.56
D2	58.87	68.79	39.13	68.79	37.5	56.25
D3	75.25	79.29	62.16	79.29	55.2	52
D4	79.51	81.15	82.98	81.15	69.64	71.43
D5	96.26	93.05	100	93.15	94.44	89.68
D6	96.92	92.31	100	92.31	95.2	88
D7	73.88	76.69	60	76.69	50.89	42.86
D8	90.38	90.38	98.98	90.38	88.24	88.24
Average	80.38	82.34	78.02	82.34	71.39	71.75
Max	96.92	93.05	100	93.15	95.2	89.68
Min	58.87	68.79	39.13	68.79	37.5	42.86
Median	77.93	80.22	81.94	80.22	74.82	78.495

On average, Ensemble KNN achieves a higher accuracy (82.34%) compared to Classic KNN (80.38%). The highest accuracy for Classic KNN is 96.92% (D6), while for Ensemble KNN, it is 93.05% (D5). However, the minimum accuracy of Classic KNN is significantly lower (58.87% for D2) compared to Ensemble KNN (68.79% for D2), indicating that the ensemble approach provides more stable and improved performance, particularly for datasets where Classic KNN struggles.

Precision measures the proportion of correctly classified positive cases. Here, Ensemble KNN shows better overall performance, with an average precision of 82.34% versus 78.02% for Classic KNN. The maximum precision for Classic KNN reaches 100% (D5, D6), but this is not necessarily an advantage if recall is low. Ensemble KNN maintains higher precision for most datasets, reflecting its improved ability to classify positive instances correctly.

Recall, which measures how well the model captures all actual positive instances, is slightly more balanced between the two methods. Ensemble KNN achieves a slightly higher average recall (71.75%) compared to Classic KNN (71.39%). The maximum recall for Classic KNN is 95.2% (D6), while for Ensemble KNN, it is 89.68% (D5). However, Ensemble KNN reduces the minimum recall (42.86% vs. 37.5%), demonstrating better handling of difficult classification cases.

Ensemble KNN generally improves accuracy and precision while maintaining a more stable recall across datasets. This suggests that ensemble techniques help mitigate Classic KNN's weaknesses, particularly in challenging datasets. Table 2.3 provides a summary of comparison between the different types of classification algorithms.

Table 2.3 Classification Algorithms.

#	Classification Algorithm	Description	Strengths	Limitations
1	**K-Nearest Neighbors (KNN)** [37]	A simple, instance-based learning algorithm that classifies data based on the majority label of its k nearest neighbors.	Simple to understand and easy to implement. Non-parametric (does not assume any underlying distribution). Effective with small to medium datasets. No training phase, as it is based on memorization.	Computationally expensive during inference, as it requires calculating distances to all points. Sensitive to irrelevant features and noise.
2	**Support Vector Machine (SVM)** [98]	A supervised learning algorithm that finds the hyperplane that best separates data into classes.	Works well with high-dimensional data. Effective in cases where the number of dimensions exceeds the number of samples. Can handle both linear and non-linear classification tasks using kernel tricks.	Memory-intensive. Sensitive to the choice of kernel and parameters. Does not perform well with large datasets.
3	**Logistic Regression** [99]	A probabilistic classification model that estimates the probability of an instance belonging to a class.	Efficient for binary classification tasks. Works well when the decision boundary is linear. Interpretable model (coefficients represent feature importance).	Assumes linear relationships between features and class labels. Performs poorly with non-linear decision boundaries.
3	**Decision Trees** [77]	A hierarchical model that splits the data into subsets based on feature values, leading to decision nodes	Easy to interpret and visualize. Can handle both categorical and numerical data. Handles missing values well.	Prone to overfitting. Sensitive to small variations in data. Can be unstable if not pruned.
4	**Random Forest** [53]	An ensemble method that combines multiple decision trees to improve classification accuracy and generalization.	Reduces overfitting compared to individual decision trees. Handles large datasets well. Less sensitive to outliers.	Computationally expensive. Less interpretable compared to individual decision trees
5	**Naive Bayes** [20]	A probabilistic classifier based on Bayes' theorem, assuming independence between features.	Simple and fast. Performs well with a small number of data. Works well with text classification and high-dimensional data.	Assumes that features are independent, which is often unrealistic. Poor efficiency with correlated features.

KNN is a great match for DBSCAN in improving fog computing performance because it's easy to use, works well with other tools, and is effective in real-time situations. While DBSCAN does a great job at spotting outliers and grouping data with varying densities, it doesn't have a built-in way to classify that data [23]. By combining DBSCAN with KNN, we get the best of both worlds DBSCAN helps organize the data by finding meaningful clusters, and KNN can then classify individual data points based on how close they are to those clusters.

In fog computing environments, where resources are limited, KNN performs well because it's a simple, non-parametric method that doesn't require a lot of training [100]. Since KNN classifies data based on distance, it works even better when the data is already well-organized something DBSCAN helps with by grouping similar points. This combination is especially useful in dynamic and decentralized IoT systems, where quick and accurate decisions need to be made in real time [21].

Furthermore, KNN's capacity to generate local predictions using the closest neighbors fits in nicely with fog computing's edge based architecture, which lessens reliance on centralized cloud systems [73]. The best way to enhance fog efficiency in time sensitive Internet of Things applications is to combine the powerful clustering of DBSCAN with the quick and efficient classification of KNN.

2.9 Fog Simulation Platforms

Applications in fog and IoT contexts may be mimicked with event-based emulator. This provides a framework for evaluating resource management strategies with a number of metrics, like energy operating costs, consumption, and network congestion [101]. Simulators are provided for several distributed contexts, including cloud, grid, and fog edge. FogTorch, EmuFog, EdgeCloudSim, FogNetSimCC, iFogSim, iFogSim2, and YAFs are some of the simulators used for fog computing.

The simulators were grouped by [60] based on the criteria, considered critical for realistic modelling of fog situations. The topology's structure is the initial prerequisite. The topology allows for the representation of the network's infrastructure. A tree-based structure is used by EdgeCloudSim, iFogSim, and iFogSim2. Conversely, a

graph structure is used by YAFS, EmuFog, FogTorch, and FogNetSimCC. YAFS also allows definition of subgraphs for a topology.

Subparagraphs are used by [43], to denote isolated colonies. The multiple components and linkages involved in creating topologies make the process difficult and time consuming. The code for the scenario is the subject of the second requirement. While some fog simulators allow you to create the properties of the scenario through an API, YAFS allows definition of the scene using JSON based files. Syntax may as well be used to specify the topology [102].

The FogNetSim is accessed through the graphical user interface (GUI). One of the key features to look at is how the simulator handles and presents results. When advanced scenarios with customizable policies are created, it's important that every event in the simulation is saved into files [92]. This makes it easier for users to analyze the data in depth and detect specific performance patterns. Another important requirement is the ability to change fog computing conditions while the simulation is still running. To accurately reflect real-world situations, simulations must account for changes in various factors, such as user mobility. Both EdgeCloudSim and FogNetSimCC allow infrastructure adjustments that mimic users moving around [1].

While most simulators are developed in Java, YAFS and FogNetSimCC stand out by being written in Python and C++, respectively. FogTorch, another simulator, uses Monte Carlo simulations to find the best way to allocate applications based on Quality of Service (QoS) metrics like latency, cost, bandwidth, and response time [96]. It focuses specifically on solving the challenge of deciding where to deploy applications.

A good fog simulator should make it easy to model different scenarios and applications, run simulations quickly, and deliver accurate performance metrics such as latency and energy consumption [64]. Past research in cloud simulation has shown that different tools offer different strengths and trade-offs [6]. For this study, the YAFS simulator will be used to evaluate the hybrid model.

2.10 Paradigm of Synthetic Dataset

Synthetic data, created using algorithms or simulations to mimic real world data has become an essential tool, especially in areas like machine learning where privacy concerns, high costs, or limited access to real data can be an issue [89], [90]. Unlike real datasets, synthetic data gives researchers more control, makes it easier to simulate rare or unusual scenarios, and avoids the legal and ethical challenges that come with using personal information [97]. In fog computing, where real-world data can be hard to obtain, synthetic data serves as a valuable and practical alternative for testing models and improving their performance.

2.10.1 Synthetic Dataset in Machine Learning

Several studies demonstrate the utility of synthetic datasets in evaluating feature selection methods such as Random Forest and Recursive Feature Elimination (RF-RFE). For instance, [96] employed synthetic data to assess the efficacy of RF-RFE in identifying critical variables, validating its scientific reliability. Similarly, [98] and [99] applied the RF-RFE method to synthetic data to test the robustness and accuracy of feature ranking in controlled conditions. These studies prove that synthetic datasets can effectively isolate the influence of individual features, making it easier to interpret their impact on model performance.

[100] further validated the use of synthetic data in high-dimensional scenarios, confirming its role in enhancing classification accuracy and model reliability. Additionally, [101] emphasized how synthetic datasets facilitate deeper analysis by allowing varied interaction scenarios between variables. For fog computing, synthetic datasets enable complete data coverage, controlled experimentation, and avoidance of missing data issues. By designing datasets that include relevant metrics like latency, throughput, and packet loss, researchers can tailor data precisely for RF-RFE investigations, ensuring reproducible and interpretable results.

2.10.2 Case Studies on Synthetic Dataset

A number of case studies have shown the effectiveness of synthetic data across a variety of domains. One prominent study by [98]examined the use of synthetic data in predicting blood pressure, comparing machine learning models trained on real data,

synthetic data, and a combination of both. Their findings showed no significant difference in efficiency across these datasets, suggesting that synthetic data can effectively train models with accuracy comparable to real data. In recent years, the application of synthetic datasets has gained significant traction in evaluating machine learning models, addressing issues like data scarcity, privacy concerns, and the requirement for diverse training data [98].

Author [103] presented SynEval, an open-source evaluation framework for evaluating the fidelity, usefulness, and privacy preservation of synthetically created tabular data in the field of natural language processing. The efficiency of the framework in assessing the quality of synthetic data and its usefulness for subsequent tasks were demonstrated when it was used to synthetic product review data produced by cutting-edge big language models.

Additionally, explainable machine learning has advanced thanks in large part to synthetic datasets. The Two4Two project was created by [86], who introduced a synthetic dataset intended for controlled tests to assess interpretability techniques in picture classification. This method helps create more transparent AI systems by enabling researchers to methodically evaluate and contrast various explanation strategies.

Additionally, [88] suggested XAI-Bench, a library and set of synthetic datasets for feature attribution algorithm benchmarking. The creation of synthetic data that replicates real-world patterns allows researchers to assess feature attribution techniques more effectively, which advances the creation of interpretable AI models.

To improve machine learning model efficiency in the automotive sector, especially when real data is hard to come by or scarce, synthetic datasets have been used. For autonomous driving (AD) and advanced driver-assistance systems (ADAS), Applied Intuition provides synthetic datasets that increase model accuracy, speeding up training and lowering expenses related to data collection and labeling.

Despite these advancements, the reliance on synthetic data has sparked debate within the AI community [104]. Synthetic datasets have proven to be valuable assets in

evaluating and enhancing machine learning models across various domains. These case studies demonstrate that, when used judiciously, synthetic data can effectively supplement real data, leading to improved model efficiency and facilitating advancements in AI research.

2.11 Variables Determining Fog Efficiency

Using the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) criteria, 97 articles were retrieved based on specific key words from five standard databases. Inclusion criteria was followed in the retrieval and exclusion criteria was applied to select only 30 articles for the review as described in section 3.3.1. The reviewed articles were categorized based on the primary factors featured. Purposively, five articles were selected for each identified variable, as summarized in table 2.4.

Table 2.4: Variables Determining Fog Efficiency.

Variable	Authors
Latency	[88] [58] [64] [55] [17]
Channel Utilization	[107] [108] [88] [63] [109]
Throughput	[110] [40] [37] [81] [41]
Response Time	[9] [111] [37] [63] [37]
Packet Loss Rate	[38] [39] [112] [34] [68]
Queue Time	[9] [88] [63] [73] [113]

Each of the identified variables is elaborated to provide a clear understanding on how it relates with fog computing. Case studies of its implication to efficiency are cited and elaborated.

2.12.1 Latency

In fog computing, latency the delay in how quickly data is transferred or processed can be grouped into three main categories: network latency, processing latency, and communication latency. Network latency is the delay caused when data moves between devices, and it's often affected by things like network congestion and the quality of the network infrastructure [112].

Processing latency depends on how powerful the fog node is, how complex the algorithm is, and how busy the system is, all of which can slow down real-time

decision-making [113]. Communication latency happens when data is exchanged between fog nodes and cloud servers, or between fog nodes themselves, and is influenced by both the network's condition and the communication protocols in use [34].

High latency can have a serious impact on real time applications like autonomous vehicles, smart grids, and virtual or augmented reality systems [8]. For instance, delays in processing sensor data in self-driving cars can threaten safety, while lag in VR or gaming leads to a poor user experience [7]. Even though fog computing is designed to process data locally and reduce dependence on centralized servers, significant delays at fog nodes can cancel out those benefits—lowering performance and making the system less scalable [6][8].

Studies show that keeping latency low is crucial in smart cities, self-driving cars [114], and immersive technologies like AR/VR [23] [64]. Achieving this requires efficient local data processing and smart communication protocols that cut down delays. [61] Studied how latency affects the overall efficiency of fog computing by focusing on strategies that make smart decisions about when and where to offload tasks. They highlight that high latency slows down how fast data is processed and responded to, which is a major issue in real-time systems.

Their approach includes optimizing task placement using real-time latency measurements, which helps speed up critical operations and makes the fog layer more responsive and effective. [100] also explored ways to reduce latency in fog computing. They pointed out that high latency hurts system performance by slowing down data processing and extending response times.

In their research, they reviewed several methods to cut latency like improving data routing and boosting computing power at the fog level. Their findings suggest that when latency is effectively managed, the fog layer becomes faster, more reliable, and better equipped to handle Internet of Things (IoT) applications.

2.12.2 Chanel Utilization

[107] investigate the effect of channel use in the efficiency of fog computing systems. They define channel utilization as the rate at which network channels are used to transfer data. Their findings underscore the importance of optimal channel use in optimizing fog layer performance, as it has a direct impact on data throughput and overall system efficiency. Poor channel utilization can result in unused network resources, higher congestion, and slower data transfer rates.

The authors propose methods for increasing channel usage, such as refining network protocols, balancing data loads, and improving resource allocation mechanisms at the fog layer. By increasing channel utilization, fog computing systems can improve performance, support higher data transmission rates, and provide more efficient data handling for IoT applications. [108] discuss the relevance of channel utilization in fog computing systems and its impact on system performance.

They emphasize that good channel use is critical for increasing data throughput while reducing delays. Inefficient channel utilization can lead to greater network congestion, higher latency, and lower overall system performance. The research looks into methods to improve channel utilization, such as dynamic channel allocation, load balancing, and adaptive network management techniques. Fog computing systems can handle higher volumes of data more efficiently, minimize congestion, and increase the responsiveness of IoT applications by optimizing the use of network channels.

Channel utilization significantly influences fog computing performance, as it determines how effectively a communication channel is used for data transmission. According to [111], key factors affecting channel utilization include bandwidth, traffic volume, signal quality, and interference. Higher bandwidth supports increased data throughput, while excessive network traffic can lead to congestion and inefficiency. Efficient data transmission protocols and optimal fog node placement also improve channel usage. Enhanced channel utilization directly boosts data throughput [64], helping ensure timely and efficient communication.

Efficient channel usage reduces network congestion and latency, which is crucial for maintaining system responsiveness, especially in real-time applications [101]. Conversely, poor utilization causes bottlenecks and delays, negatively impacting performance. Effective resource management in fog systems heavily depends on optimized channel utilization, which eases the burden on network resources and supports scalability [117]. As fog networks expand, managing channel usage becomes even more critical to maintain consistent performance [10].

Case studies show that optimized channel utilization benefits various domains. In smart cities, it ensures responsive traffic and environmental monitoring systems [9]. In industrial IoT, it enables reliable real-time process control [11]. In healthcare, effective channel use is vital for timely telemedicine and patient monitoring, improving application efficiency and patient outcomes [10].

2.12.3 Throughput

Throughput in fog computing is vital for system efficiency and can be viewed in terms of data transmission between fog nodes and cloud servers, as well as data processing rates within fog nodes [118]. High throughput ensures timely data handling and optimal system responsiveness. [34] Identifies several factors influencing throughput, including communication channel capacity, bandwidth, processing power of fog nodes, and network traffic. Complex or large datasets can further reduce throughput due to increased processing and transmission times.

[79] emphasizes that throughput directly impacts fog system efficiency by enabling fast data handling, which is essential for applications like real-time analytics, sensor networks, and video surveillance. Higher throughput improves system responsiveness, crucial for time-sensitive operations such as autonomous vehicles and industrial automation. It also enhances network efficiency by minimizing congestion and packet loss. As fog systems scale, managing throughput becomes increasingly important for maintaining performance and supporting broader applications.

Case studies highlight the significance of throughput in various sectors. In smart cities, high throughput improves traffic and environmental monitoring [10]. Industrial IoT

benefits from improved responsiveness through optimized throughput [9]. In healthcare, enhanced throughput ensures fast, reliable data transmission for telemedicine and patient monitoring, boosting application effectiveness [11].

[110] examines how throughput affects fog computing system performance. Throughput is the rate at which data is processed and transferred via the system. Their study shows that in order to ensure that fog computing systems can effectively support real-time applications, high throughput is necessary for managing massive volumes of data. They clarify that low throughput might result in higher latency, data processing bottlenecks, and general system inefficiencies.

The authors suggest a number of optimization techniques to increase throughput, including bettering data processing algorithms, allocating network bandwidth more effectively, and streamlining resource management at the fog layer. Fog computing systems can increase their throughput by taking care of these issues, which improves their capability to handle large data streams and improve overall performance.

[114] investigates how throughput affects fog computing system performance, concentrating on how it affects data processing effectiveness. Throughput is defined as the quantity of data that is successfully transferred across a network or processed by the system in a specific length of time. According to their analysis, throughput has a direct impact on the system's capacity to manage data flow and support real-time applications, making it a crucial efficiency indicator.

Inadequate throughput can result in longer wait times, less effective systems, and worse overall performance. The authors propose many methods to boost throughput, such as bettering resource allocation tactics, boosting data processing capabilities, and streamlining network protocols. Fog computing systems can accomplish this by optimizing certain variables.

2.12.4 Response Time

Response time in fog computing refers to the duration between a client request and the reception of a response, including data transmission, processing, and return [34]. Low

response time is essential for timely task completion and the effectiveness of real-time applications. According to [72], factors influencing response time include network latency affected by congestion, distance, and infrastructure quality as well as the computational power of fog nodes, data size, processing algorithm complexity, and protocol efficiency. Additionally, the current load on a fog node, including active tasks and resource availability, plays a key role.

[81] Highlights that response time directly affects the performance of real-time applications such as autonomous vehicles, smart grids, and augmented reality. High response time can delay critical decision-making, reducing operational safety and system reliability. Efficient response time ensures rapid data processing and action, improving resource utilization and increasing system throughput.

Response time is also closely tied to user experience in applications like VR, gaming, and streaming, where high response times cause lag and reduce engagement. As fog systems expand to support more devices and services, maintaining low response time becomes more challenging. However, doing so is vital for promoting system scalability and maintaining performance across diverse and growing fog computing environments. [9] Emphasizes the importance of response time in fog computing systems and its impact on overall performance.

They explain that reaction time, or the entire time required to process and respond to a request, is a critical factor in system efficiency and user satisfaction. High reaction times can cause delays in application performance, compromising real-time data processing and interaction. Their research identifies several factors that contribute to longer response times, including latency, processing delays, and network congestion.

To increase response times, the authors suggest numerous solutions, including maximizing computer resources, improving data routing protocols, and applying efficient scheduling algorithms. By addressing these variables, fog computing systems can achieve faster response times, improving efficiency and providing a better user experience for IoT apps.

[111] investigates the effect of response time on fog computing performance, with an emphasis on the fog layer's efficiency and efficacy. They argue that response time is critical for maintaining the efficiency of fog computing systems because it directly determines how quickly data requests are processed and addressed. High response times can damage system performance, lower throughput, and provide inferior user experiences, particularly for real-time applications.

Various techniques are recommended to improve response times, including better resource allocation, improved data processing algorithms, and improved network connection. Fog computing solutions can handle real-time data processing more effectively and provide more responsive and reliable efficiency for IoT applications by lowering response time.

2.12.5 Packet Loss Rate

Packet loss rate, defined as the ratio of lost packets to total transmitted packets, plays a crucial role in fog computing performance, affecting data transmission, processing efficiency, and system reliability [120]. Minimizing packet loss is essential for maintaining high system performance. According to [96], key contributors to packet loss include high network traffic, congestion, wireless interference, faulty network hardware, and data transmission errors. Network design, particularly fog node placement, also influences packet loss rates.

As stated by [105], high packet loss can compromise data integrity critical for applications like sensor networks and real-time analytics. Lost or corrupted data reduces reliability and requires retransmissions, increasing delays and lowering throughput. This degrades network efficiency and overall system stability. Packet loss also consumes additional bandwidth and compute resources due to repeated data transmission, reducing the effectiveness of fog systems.

Real-time applications such as autonomous vehicles, video surveillance, and interactive services are particularly vulnerable to packet loss, which can degrade performance and user experience. Case studies show that in smart cities, high packet loss reduces sensor data reliability, mitigated through congestion control and error

correction [11]. In industrial IoT, packet loss affects accuracy and responsiveness [10], while in healthcare, it disrupts timely data transfer for telemedicine and monitoring [9].

[38] Investigates how packet loss rate influences the efficiency of fog computing systems. They explain that packet loss, or inability to transport data packets from source to destination, has a considerable impact on the fog layer's effectiveness. High packet loss rates can cause incomplete data transfers, reducing the dependability and quality of service provided by fog computing systems.

The authors explain how packet loss can result in data re-transmissions, higher delay, and reduced throughput, all of which contribute to overall system inefficiencies. Their findings highlight the significance of adopting strong error correction and packet recovery techniques to limit the negative consequences of packet loss. By mitigating packet loss, fog computing systems can improve data integrity, reduce re-transmission costs, and increase overall performance.

[39] Investigates the effect of packet loss rate on the efficiency of fog computing systems, with a focus on data management and system efficiency. Their findings show that high packet loss rates can contribute to longer retransmission times, lower data throughput, and overall system inefficiencies. They examine several solutions for reducing packet loss, such as enhanced error detection and correction techniques and enhancing network protocols to provide more reliable data transfer. By addressing packet loss, the fog layer can increase performance, as less packet loss leads to more effective data handling and system responsiveness.

2.12.6 Queue Time

Queue time in fog computing refers to the duration a data packet or task spends waiting in a queue before being processed by a fog node. This metric is influenced by task arrival rates, fog node processing power, and queue management policies, as noted by [73]. Effective queue management is essential for ensuring timely data processing and maintaining overall system efficiency. According to [87], factors affecting queue time include high task arrival rates, limited processing capacity, network latency, and the

fog node's workload. Queue management strategies such as FIFO, Priority Queuing, and Round-Robin also play a significant role in determining queue times.

Extended queue times negatively affect data processing efficiency, especially in real-time IoT applications like video analytics, sensor monitoring, and smart control systems. Delays reduce throughput and responsiveness in systems requiring rapid decision-making, such as autonomous vehicles, smart grids, and industrial automation. Additionally, prolonged queue times can cause network congestion and inefficient resource utilization, reducing overall performance.

Case studies highlight that smart cities require efficient queue management to process large sensor data volumes effectively [28]. In industrial IoT, queue time affects timely data processing and control [10]. In healthcare, reducing queue time ensures responsive telemedicine and patient monitoring systems [12], enhancing overall application reliability and performance.

[115] investigates the effect of queue time on fog computing system performance. They define queue time as the time it takes for jobs or data packets to be processed once they are placed in a queue. According to the authors, long queue times can increase latency, limit system throughput, and degrade overall efficiency in the fog layer. Queue time has a direct impact on the efficiency of task execution and data handling because high wait times can slow down the entire processing pipeline.

Their research stresses the importance of efficient queue management systems for reducing queue time, such as dynamic scheduling algorithms and load balancing techniques. By minimizing queue time, fog computing systems can increase their responsiveness, throughput, and overall performance, allowing them to better handle real-time IoT applications. [87] investigate the effects of queue time on the efficiency of fog computing systems, with a focus on data processing and system efficiency.

They emphasize that excessive queue time can cause delays in data processing, longer response times, and lower overall system throughput. Their analysis identifies several variables that contribute to long queue times, including resource contention and ineffective scheduling. To address these challenges, the authors present a number of

queue time optimization options, including sophisticated scheduling algorithms and improved resource allocation. By reducing queue time, the fog layer can improve performance, resulting in faster data processing and more effective handling of IoT workloads.

2.13 Research Gaps

The efficiency of the fog layer is critical to the success of Internet of Things (IoT) systems because it directly determines how well data is processed, transmitted, and acted upon in real time. Unlike cloud computing, where delays can hinder responsiveness, fog computing operates closer to end devices, enabling faster decisions and improved user experiences. However, the ability of fog nodes to deliver these benefits depends largely on how well key performance variables are managed. Metrics such as latency, throughput, packet loss, queue time, and channel utilization play central roles in defining system performance.

For instance, reducing latency ensures rapid responsiveness, while higher throughput allows more data to be processed effectively. Similarly, minimizing packet loss and queue time prevents bottlenecks, and optimizing channel utilization guarantees that communication pathways are used efficiently. Despite their importance, current research often treats these variables independently, without examining their relative weight in contributing to overall fog efficiency. Some models attempt to improve one or two parameters but sacrifice others, leading to fragmented solutions that cannot scale effectively in dynamic IoT environments.

This creates a critical gap: the absence of a clear, ranked understanding of which variables most significantly influence fog performance. Without such prioritization, optimization models risk focusing on less impactful factors, resulting in wasted resources and reduced efficiency. A few attempts, such as applying machine learning to rank parameters, have been made, but these approaches have not been fully integrated into comprehensive frameworks for improving fog performance.

Moreover, the role of data quality and accurate classification has been largely overlooked, even though unreliable or poorly categorized data undermines decision-

making in areas such as task offloading and anomaly detection. This study aims to address these gaps by identifying, ranking, and integrating key performance variables into a robust model that enhances fog efficiency while maintaining accuracy across diverse IoT applications.

2.14 Conceptual Framework

A conceptual framework provides a structured approach to identify, organize, and analyse the key factors affecting system efficiency [116]. The relationship between independent, dependent and moderating variables is illustrated in figure 2.3, intended to enhance the reliability and reproducibility of study findings, enabling targeted optimizations in fog computing environments.

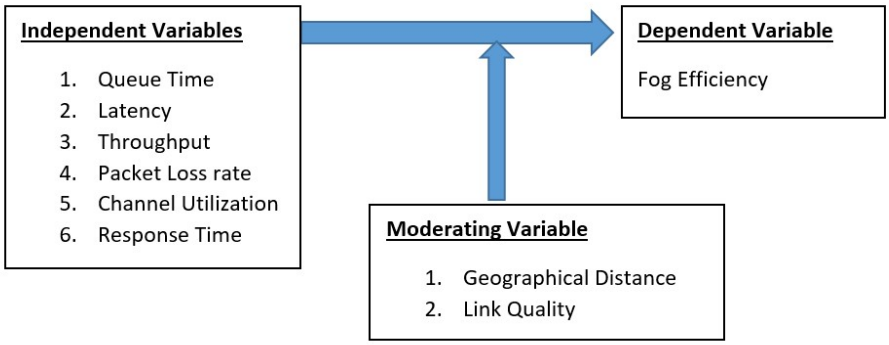


Figure 2.3: Conceptual framework

The interaction between the independent variables with the dependent variable is dynamic. For instance high channel utilization and low latency generally improve throughput and reduce response time, enhancing overall fog efficiency. Conversely, increases packet loss and prolonged queue time typically degrade performance. Each variable contributes differently, with latency and packet loss having immediate impacts on real-time data processing, while throughput reflects longer term system capacity.

Moderating variables, geographical distance and link quality influence these relationships. For instance, greater distance can increase latency and packet loss due to signal attenuation and propagation delay, weakening the efficiency of fog services. Similarly, poor link quality can disrupt data transmission, elevating queue time and reducing throughput. These moderating variables do not act independently but shape

the strength and direction of the impact that each independent variable has on fog efficiency, making their consideration vital for accurate modeling.

2.15 Chapter Summary

This chapter presents the theoretical grounding of fog computing based on what is available in literature. The IOT architecture is discussed, which lays foundation of the succeeding sections. Justification of use of the word fog computing is done from where the other aspects of the research title are discussed. Fog computing enhancement techniques are discussed in detail, which lay foundation for the succeeding chapters. This includes explanation and citation of related hybrid techniques and approaches used to solve fog efficiency challenges.

Both clustering and classification algorithms are compared, citing their strengths and weaknesses, which is a justification of the choice of algorithms to make the hybrid model under investigation. Different algorithms and methods used in literature are discussed. Fog simulation techniques are discussed and comparison of various simulation platforms is presented.

A summary of techniques proposed by various authors is presented, as well as hybrid models for addressing fog performance. Their achievements and limitations are pointed out. A systematic literature review is carried out to identify the variables which determine fog performance. The variables are discussed to highlight their significance and case studies.

CHAPTER THREE
METHODOLOGY

3.1 Introduction

This chapter presents the methodology that guided the development and evaluation of the proposed hybrid model aimed at improving fog computing performance in the Internet of Things (IoT) environment. It outlines the overall research approach, starting with the guiding research philosophy and design strategies aligned with the study's four objectives. Each objective is addressed through a distinct design framework, tailored to capture relevant data and insights. The chapter also details the target population and sampling strategy used to select suitable data sources for variable validation.

Data collection instruments are discussed, along with the procedures used to ensure their validity and reliability, including the implementation of a pilot study. In addition, the experimental materials used in the simulation and modeling phases are outlined, followed by a description of the data analysis methods employed to interpret and present the results. Finally, the chapter emphasizes ethical considerations that were observed throughout the research process, ensuring integrity and transparency. Overall, this methodology provides a structured, reliable, and ethical approach to answering the research questions, validating the hybrid model, and contributing valuable insights to the field of fog computing and IoT systems.

3.2 Research Philosophy

This study is based on the philosophy of pragmatism, which takes a practical and flexible approach to understanding truth and knowledge. Instead of viewing truth as something fixed, pragmatism sees it as changeable, depending on what works best in a given situation. According to this perspective, truth is not about being universally correct, but about being useful and effective in solving real world problems. In this sense, ideas are treated as working theories. They are considered valid if they produce results when tested in practice or through experience.

Pragmatism is a good fit for this research because the study focuses on real life challenges in fog computing, such as reducing latency, packet loss, and congestion. The philosophy supports the use of both qualitative and quantitative methods, which aligns with the study's mixed method approach: identifying and ranking performance variables, developing and testing a hybrid model, and validating it through simulations and expert reviews.

It also gives the researcher the flexibility to choose the most appropriate tools and methods like data driven algorithms (RF-RFE, DBSCAN, KNN) and simulation tools (YAFS) to address the research problem. Pragmatism matches well with engineering and computing research, where results, efficiency, and relevance often matter more than sticking to a single way of thinking.

The assumptions guiding the study philosophy include:

i. The study assumes that the variables it identifies are the key factors that influence how well fog computing works, and that the ranking method used will clearly show which of these factors matter most.
ii. It is assumed that the experts participating in this study will have a similar level of knowledge and experience in fog computing, making their input consistent and reliable.
iii. Combining different algorithms will create a more effective solution by leveraging the strengths of each.
iv. The model developed in this research is expected to lead to noticeable improvements in fog computing performance, such as faster processing and fewer delays.
v. It is assumed that the results from the simulation will accurately show how well the model performs in a real fog computing environment.

The study emphasizes practical outcomes through the integration of theory and application. It supports real world decision making by combining DBSCAN and KNN for effective data handling. The implications include enhanced system responsiveness, improved resource allocation, and data driven optimization in fog environments, offering flexible, context aware solutions that align with dynamic IoT demands and stakeholder needs.

In this study, both qualitative and quantitative data collection methods are applied to provide a holistic understanding of fog layer efficiency. Quantitative data is primarily gathered through structured questionnaires administered to industry experts. The questionnaire focuses on ranking critical performance variables such as latency, throughput, packet loss, queue time, and channel utilization.

This structured format ensures that measurable, numerical data can be obtained for statistical analysis. Descriptive statistics, mean rankings, and inferential methods are used to determine the relative importance of each variable, while reliability tests such as Cronbach's Alpha ensure internal consistency of responses.

On the other hand, qualitative data is collected through open-ended sections of the questionnaire, where respondents provide additional insights on other factors influencing fog performance, as well as their professional experiences in handling IoT environments. These qualitative responses are analysed using thematic analysis, enabling the study to identify patterns, perspectives, and emerging variables not captured in the closed-ended questions.

Combining both data types allows the study to not only rank and quantify the importance of performance variables but also to capture expert opinions and contextual details that deepen understanding. This integrated approach strengthens the validity and richness of the findings

3.3 Research Design

This study uses Design Science Research (DSR**)** as a guiding approach to develop practical solutions for real-world challenges in fog computing. The process revolves around building, refining, and evaluating innovative IT tools known as artifacts that improve performance in fog environments. In the first objective, DSR is applied by identifying and ranking the most critical variables affecting fog performance using the Random Forest Recursive Feature Elimination (RF-RFE) algorithm. These results are then validated with feedback from industry experts to ensure their relevance and accuracy.

In the second objective, DSR drives the creation of a custom algorithm that merges DBSCAN (used for clustering and filtering out noise) with KNN (used for classifying data), forming a solution specifically tailored to improve fog efficiency. The third objective involves development of a hybrid model using this algorithm, focusing on solving key performance issues in fog environments demonstrating both practical value and academic rigor. Finally, the fourth objective evaluates the model using YAFS, a simulation tool that tests how well the model works in realistic fog settings, providing evidence of its usefulness an essential aspect of Design Science.

The research takes a mixed methods approach, blending quantitative techniques like machine learning and simulations with qualitative expert feedback. This combination ensures that the technical aspects are grounded in real world insight. The overall methodology flows through three key steps: identifying important variables, developing a smart algorithm, and testing the model's performance an approach that's both systematic and solution focused. The study follows the workflow shown in figure 3.1

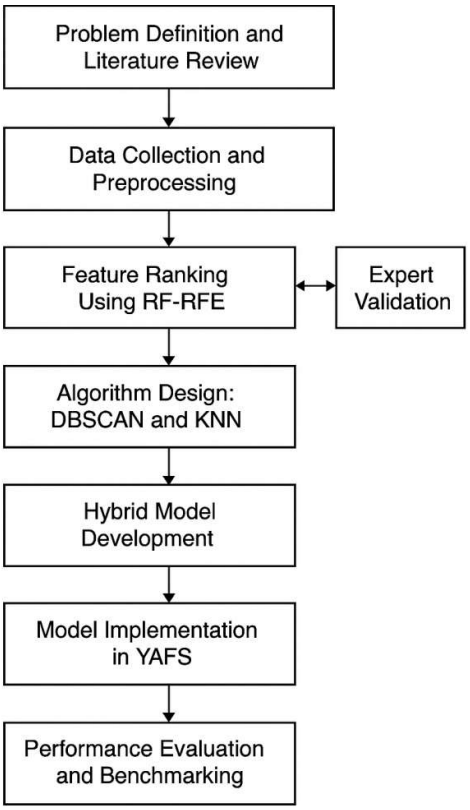


Figure 3.1 Methodological workflow for the hybrid model.

This workflow balances quantitative and qualitative methods by combining machine learning with expert validation and simulation. The RF-RFE step ensures informed feature raking, while DBSCAN removes noise before KNN classification improves prediction accuracy. The YAFS simulation allows for testing under varied fog computing conditions, enhancing practical relevance and model generalizability [117], [118].

3.3.1 Research Design for Objective One

To accomplish the first objective, an exploratory research design is used. A systematic review is used to evaluate the important variables that influence fog efficiency in IoT applications. To achieve this, PRISMA criteria is used to review articles published between 2020 and 2024. The searched databases were; Springer Link, IEEE Xplore, Science Direct, and ACM. The key words used in the search were; “IoT performance”, “Fog variables”, “fog efficiency”, “Fog computing” and “Fog optimization”.

A total of 97 articles are retrieved and an exclusion criteria is applied where all the irrelevant, duplicate studies, and non-peer-reviewed articles are discarded. Further screening of the articles was done by reviewing the abstracts, out of which a total of 30 papers were selected for final review.

To rank the variables, Radom Forest Recursive Feature Elimination algorithm (RF-RFE) was used. Secondary synthetic dataset was used for training and testing. After this, an exploratory approach was used to engage industry experts to validate the ranking based on their experience and expertise on the subject matter.

3.3.2 Research Design for Objective Two

The second objective involved designing an algorithm to improve fog performance, the study uses a design science research approach**.** This method focuses on building a practical and innovative solution by combining two well-known machine learning techniques: DBSCAN and KNN. DBSCAN is used to identify clusters and filter out noise in the dataset, while KNN helps classify the data points based on their proximity to these clusters. The design process begins by understanding the unique strengths of

each algorithm and how they can complement each other. The goal is to create a hybrid that handles noisy, real-world fog data more efficiently than using each method alone.

3.3.3 Research Design for Objective Three

To achieve Objective Three, the study uses a combination of descriptive and experimental research designs**.** The descriptive approach is used to guide the development of a model aimed at improving fog performance. This involves detailing the components and structure of the model based on insights gained from the previous objective. Once the model is fully developed, the research shifts to an experimental design to test how well the model works in practice. This blended design approach allows the study to both explain the model's design in detail and rigorously evaluate its practical utility, ensuring the solution is both theoretically sound and applicable in real world fog computing scenarios.

3.3.4 Research Design for Objective Four

To achieve Objective Four, the study uses a mix of experimental and exploratory research designs. This stage focuses on evaluating how well the hybrid model performs when applied in a virtual fog computing environment. Using a fog simulation platform, the model was tested through a series of experiments designed to reflect real-world conditions. The experimental design helps in systematically measuring the model's performance, while the exploratory approach allows for a deeper investigation into how the model behaves under different scenarios.

Key variables such as latency, packet loss, and congestion are tracked during the simulation. Additionally, statistical analysis is conducted to interpret the results, and benchmark comparisons are made to assess how the new model performs against existing solutions. This combined design helps validate whether the proposed model actually leads to improved efficiency in fog environments, making the findings more practical and relevant for real-world applications.

Table 3.1 Mapping research objectives to research design

#	Research Objective	Research Design Step	Expected Output
1	i. To analyze the variables influencing fog layer efficiency, rank them through RF-RFE, and validate with industry experts.	-Reviewing literature to identify key performance variables. -Application of Random Forest Recursive Feature Elimination (RF-RFE) for ranking. -Validating rankings with expert feedback.	Ranked and validated list of critical fog efficiency variables.
2	To design a DBSCAN and KNN based algorithm for improved fog layer efficiency.	-Formulating DBSCAN clustering algorithm for grouping tasks and managing noise. -Implementing KNN classification for decision-making.	Prototype algorithm combining DBSCAN and KNN tailored to fog computing.
3	iii. To develop a hybrid model based on the two algorithms and test its effectiveness in improving fog layer efficiency.	-Integrating ranked variables into DBSCAN + KNN framework. - Building a hybrid optimization model. -Test model performance using experimental datasets.	Hybrid model capable of enhancing fog layer efficiency in IoT environments.
4	iv. To evaluate the hybrid model using YAFS (Yet Another Fog Simulator) and compare results with benchmarks.	-Implementing hybrid model within YAFS simulation environment. -Running experiments and collecting performance metrics	Comparative analysis of model efficiency against benchmarks; validated performance improvements.

3.4 Target Population and Sampling Strategy

Objective one involves collection of primary data where industry experts are sampled to evaluate the variable ranking. The survey targeted a population of experts for ranking the variables. Purposefully the expert population was selected using criteria by [119] which were: professional training, experience of at least 2 years in IoT related areas. Snow bowling technique was used to sample 30 experts. The necessary education back ground included IOT certification on the IOT professional courses. This is because the IOT specialty is still in its early stages, and IOT experts are randomly distributed in the industry.

Statistically, the variability level of the population is very low, therefore the sample size is considered sufficient to provide accurate and reliable estimates of population parameters, to minimize sampling errors, and detect meaningful effects or relationships with adequate statistical power. By analysing this dataset, the study identified the critical efficiency variables. Machine learning and statistical methods were employed to analyse data and derive insights regarding the importance and interdependencies of these efficiency variables.

Objective two, three and four involved algorithm formulation, model development, testing and evaluation of the hybrid model, where secondary data was used. This is due to lack of datasets with all features under investigation, and that the synthetic dataset allows complete control over the dataset's characteristics ensuring well-structured and diverse characteristics. It also gives a balanced and noise controlled data, reducing bias and ensuring fair evaluation.

3.5 Data Collection Instruments for Variable Validation

A structured questionnaire was designed for data collection and administered to the industry experts for the expert opinion on the validation of the ranking of variables for fog computing performance. A pilot study was carried out to test the data collection tool. A questionnaire was administered to ten purposefully selected industry experts and Cronbach's alpha testing carried out on the collected data.

The respondents in the pilot test did not participate in the final survey, to avoid response bias and ensure data integrity. Having seen the questions before, pilot participants might approach them differently, influenced by prior exposure, which could skew the results. Using fresh participants ensured objective, reliable, and unbiased data collection for the survey.

3.6 Validity and Reliability of the Data Collection Instruments

The questionnaire was carefully reviewed by the researcher and the supervisors to look for any abnormality that could undermine the credibility of the data in gathering instruments in order to maintain the validity of the data collection instruments. The supervisors assisted in identification of key contributions by each question. They gave guidelines on translating the questionnaire to an online platform and how to analyze the data.

3.6.1 Pilot Test

Industry experts were used to validate the variable ranking. A pilot study is carried out to test the validity of a data collection tool. A questionnaire (Appendix 1) is administered to ten purposefully selected industry experts and Cronbach's alpha

testing run on the collected data. This test evaluates the internal consistency and dependability of the measuring instruments. Cronbach's alpha is statistical metric used to assess how well a group of items or indicators in a test or survey measure a particular construct.

In this case, Cronbach's alpha helps verify that the various items or questions are measuring the same underlying concept. The pilot study is carried out by administering questionnaires to ten industry experts selected randomly. The output is tabulated in table 3.1.

Table 3.1 Responses from pilot testing.

Respondents	Latency	Throughput	Channel Utilization	Queue Time	Packet Loss Rate	Response time	Total
A	3	4	4	5	4	5	20
B	3	3	4	4	4	5	18
C	4	4	4	5	5	5	22
D	5	3	4	3	4	5	19
E	5	5	5	5	5	5	25
F	5	4	4	4	5	5	22
G	5	5	5	5	5	5	25
H	5	5	4	5	5	5	24
I	5	5	5	5	5	5	25
J	4	4	5	4	5	5	22
							6.62
Variance	0.71	0.62	0.27	0.5	0.23	0.23	2.33

The Cronbach's alpha values was realized using the following formula:

$$\alpha = \left(\frac{k}{k-1}\right)\left(\frac{sy^2 - \varepsilon s_i^2}{s^2 y}\right)$$

$$= (6/5)\ ((6.62-2.33)/6.62)$$

$$= 0.8$$

$$\text{P-value} = 0.045$$

This confirms reliability of the data collection instrument and therefore its suitability for the study. Cronbach's alpha is a measure of internal consistency or reliability of a scale, used in survey. According to [42], the value of Cronbach's alpha ranges from 0 to 1, where: 0.9 and above: Excellent (High reliability). 0.8 to 0.9: Good (Acceptable reliability). 0.7 to 0.8: Acceptable (Adequate reliability). 0.6 to 0.7: Questionable

(Low reliability). 0.5 to 0.6: Poor (Unreliable). Below 0.5: Unacceptable (Very low reliability).

Generally, a higher Cronbach's alpha indicates better internal consistency, but a very high value (close to 1.0) might suggest redundancy in the items, meaning they are too similar to each other. A commonly accepted threshold for good reliability is 0.7 or higher. The data collection instrument demonstrated acceptable validity, as confirmed by its reliability through Cronbach's alpha. With a score above 0.7, the instrument shows adequate internal consistency, indicating that the survey items effectively measure the same underlying construct.

3.6.2 Tool Validity

The reliability of the data collection tool was confirmed using Kendall's Coefficient of Concordance (W). The computed value of W = 0.728656361 indicates a moderate to strong level of agreement among respondents regarding the ranking of fog computing efficiency variables. While there is some variation in their responses, the overall consistency of the rankings demonstrates substantial reliability. Furthermore, the p-value = 0.0436568, which is less than the 0.05 threshold, provides evidence of statistically significant agreement at the 5% level.

This implies that the observed concordance among respondents is unlikely to have occurred by chance, thus strengthening the credibility of the findings. In addition, the Chi-square statistic = 84.52413793 further confirms that the agreement is not random but represents a meaningful pattern of consensus. Together, these statistical measures validate the robustness and reliability of the data collection tool for assessing fog layer efficiency.

3.7 Experimental Materials

The first version of the questionnaire was used in the pilot study, and based on the feedback received, some improvements were made to develop the final version shown in Appendix 1. To rank fog computing variables and train or test the model, a synthetic dataset was used. This type of data is artificially generated to resemble real world information while ensuring it doesn't represent actual people or events. It's created

through algorithms or simulations and is designed to have the same statistical patterns as real data.

To test how well the developed model works, YAFS simulation platform was used. This tool helped recreate a realistic fog computing environment where different scenarios can be set up to evaluate the model's performance. The simulation results offered a clear picture of how much the proposed model improves efficiency by allowing comparisons across various configurations and optimization techniques in a controlled, repeatable setting.

3.8 Data Analysis

This study analyzes both quantitative and qualitative data. For the first objective, it uses a mix of exploratory and explanatory approaches, applying both descriptive and inferential statistics to validate expert feedback on key variables. In the second objective, an explanatory approach was followed, where an algorithm is created by combining two existing ones that complement each other. The third objective involves exploratory analysis along with statistical methods particularly regression and comparative analysis to evaluate and compare the outcomes of model testing. Descriptive statistics are also used to examine and visualize the results clearly.

In the fourth objective, statistical tools help assess how different experimental conditions affect model performance. These include descriptive statistics, regression analysis, and hypothesis testing. The analysis helps uncover patterns, draw insights, and make sound decisions about how well the model works. The performance is measured using variables identified earlier.

The main hypothesis in this study is that combining DBSCAN (for noise filtering) with KNN (for classification) significantly boosts the efficiency of fog computing in IoT environments more than using traditional or individual algorithm. This is tested by comparing variables like latency, throughput, and packet loss before and after applying the hybrid model in a simulated fog setup. Confirming this hypothesis supports the study's goals by showing the value of feature selection, validating the hybrid model, and demonstrating its edge in real time, decentralized decision making.

3.9 Ethical Considerations

A research approval was obtained from Murang'a University of Technology and supervisors assigned. Research approval was also sought from National Commission for Science, Technology and Innovation (NACOSTI). The subject's information was handled with confidentiality and utilized just for the study. Furthermore, participation in the survey was voluntary.

All secondary literary sources re cited accordingly. To address potential bias in the study, transparency in data collection is ensured, including sampling procedures, data sources, and data preprocessing steps. Data collection processes was well documented to identify potential sources of bias. To ensure anonymity of participants, the study used anonymous data collection methods and avoided personally identifiable data.

3.10 Chapter Summary

This chapter focuses on the methodology used to explore the specific variables that determine fog computing efficiency in IOT, where the variables are ranked using machine learning algorithm (RF-RFE), out of which industry experts are engaged to validate the ranking. The method used to develop a hybrid model is discussed. The methodology used to achieve the objectives of this study is discussed including the research philosophy. The research process is also stipulated.

The target population, sampling strategy and sample size are laid down. The instruments for data collection are discussed, including methods of testing their reliability and validity. Results of a pilot study on the data collection tool are shared confirming the validity of the instrument. Experimental materials, data analysis and ethical considerations are outlined.

CHAPTER FOUR
RANKING AND VALIDATION OF VARIABLES DETERMINING FOG LAYER EFFICIENCY

4.1 Introduction

The key fog layer efficiency determining variables have a significant impact on the fog layer's effectiveness. To make sure fog computing systems can effectively handle and analyze data from a variety of IoT devices, it is essential to recognize and modify these features [120] [121]. In addition to enhancing the functionality of specific IoT applications, an understanding of these factors increases the overall reliability and scalability of fog computing systems in a data-intensive setting.

Fog layer extends the concepts of cloud computing to the network edge by distributing processing resources over numerous nodes, including routers, switches, and local servers [3]. By processing data closer to its source, this distribution aims to improve data efficiency and decrease reaction times. Even with these developments, several challenges still exist, which affects fog computing systems' effectiveness and performance. Optimizing these systems requires an understanding of the different variables and how they affect fog performance.

4.2 Ranking of Variables

The identified variables are ranked to determine their order of significance in influencing fog performance.

4.2.1Experimental Design

This research used a carefully planned experimental design to find out which factors are most important in determining the efficiency of the fog layer in IoT. The focus was on five key variables: channel utilization, queue time, latency, packet loss rate, and throughput. Since real-world data is often difficult to collect or may not cover all scenarios, the study used a synthetic dataset that imitates real IoT traffic and fog computing conditions. This allows for full control over the variables and ensures that each one can be tested fairly.

To rank the importance of these variables, the research will apply a method called Random Forest Recursive Feature Elimination (RF-RFE)**.** Random Forest is a machine learning technique that can measure how much each variable contributes to predicting system efficiency. RFE then takes this information and gradually removes the least important variables until only the most significant ones remain, giving a clear ranking. This process ensures accuracy and reliability, showing not just which variables matter most, but also their order of importance. The outcome will guide the design of the hybrid model by highlighting which factors to prioritize in improving fog efficiency.

The platform used to carry out the experiment was a laptop computer, HP Intel(R) Core i5-6300U CPU @ 2.4GHZ, RAM 8.00 GB, and Windows 10 pro 64-bit operating system.

4.2.2 The Experimentation

Experiments were carried out to determine the order of significance for the fog efficiency variables. The RF-RFE (Random Forest–Recursive Feature Elimination) experiment is a feature selection technique used to rank variables based on their contribution to fog efficiency. The process begins with training a Random Forest model on the complete dataset, where each variable's importance is measured. RFE then recursively removes the least significant features, retraining the model at each step to evaluate performance.

This iterative process continues until only the most critical variables remain. The final ranking reflects how strongly each variable influences fog efficiency, enabling researchers to identify and prioritize factors such as latency, packet loss rate, and queue time for further optimization.

4.2.2.1 Process Flow

A Process Flow Diagram (PFD) represents the sequence of operations and data within a system. The stages are summarized in feature selection and efficiency evaluation in this particular case where Random Forest - Recursive Feature Elimination (RFE) technique is used to evaluate variables that impact efficiency in a fog layer. The

process begins with the definition of efficiency variables specific to the fog layer as summarized in Figure 4.2.

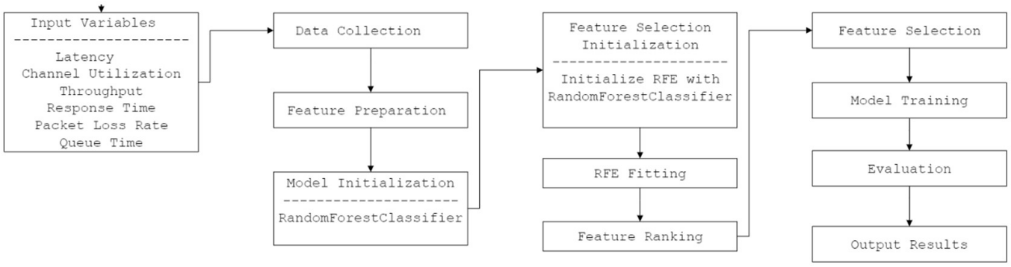


Figure 4.2: Process flow for ranking variables.

The process flow for ranking fog computing efficiency determining variables begins with data collection on key performance metrics. The data is preprocessed and fed into the Random Forest Recursive Feature Elimination (RF-RFE) algorithm. RF identifies variable importance, while RFE recursively eliminates less significant features.

Data for modelling is preprocessed. Random Forest classifier is initialized, intended for use in the top features are selected according to the ranking produced by RFE. The Random Forest model is trained using only the selected features. The model's efficiency is assessed using evaluation metrics. The final results including model efficiency and selected features is then output.

4.2.2.2 Synthetic Dataset

The first step was downloading and installation of python and pip programs. Simpy was installed using the command (Pip install simpy). Simpy was verified by opening Python Interactive Shell and typing the command (import simpy print (simpy version). Pandas was later installed through the command (pip install pandas).

4.2.2.3 Generating Synthetic Dataset

The code in appendix 2 was executed to generate synthetic dataset. This dataset was created in order to replicate IoT setups with different conditions for every variable. The dataset made it possible to validate the ranking approach and conduct controlled experiments. In this case, a Simpy environment was created, and the process of generating data was initiated. The simulation ran for the specified number of events (1000). Generated data was stored in a list of dictionaries, which is then converted into a Pandas Data Frame for easier manipulation and saved as a CSV file.

4.2.2.4 Variable Ranking Process

To rank the variables that determine fog performance, the scikit-learn library in Python was used. The dataset was split into training and test sets in the ration of 3:7 respectively. A Random Forest Regressor was instantiated for ranking the features. The RFE function was used to rank features based on their importance in predicting the target variable. The rankings were stored in a DataFrame, sorted by rank, and printed. The rankings were also saved to a csv file. Appendix 3 provides a full script for the code used to execute the ranking process.

4.3 Ranking Output

After executing the python code in appendix 3, the variables were ranked as displayed in appendix 4. A csv file was as well produced, as displayed in Table 4.1.

Table 4.1: Output of ranked variables from the most critical to the least critical

Feature	Ranking
Packet Loss Rate	1
Queue Time	2
Latency	3
Channel Utilization	4
Throughput	5

One of the variables was eliminated therefore, the initial input had six variables while the final ranked list had five variables. Fog efficiency determining variables are visualized in figure 4.3

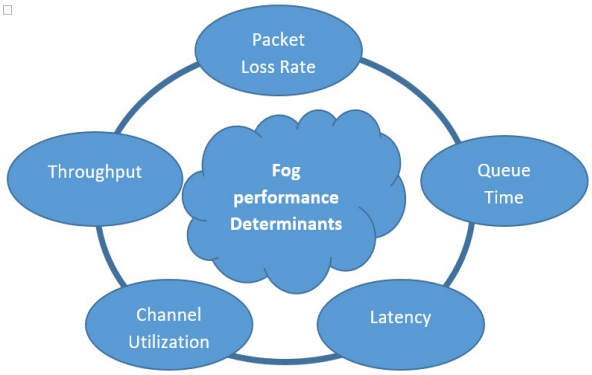


Figure 4.3: Visualization of variables determining fog efficiency.

These variables provide actionable insights into fog performance, enabling administrators to identify bottlenecks, predict potential failures, and allocate resources more efficiently.

4.4 Ranking Validation by Industry Experts

The use of industry experts to validate the ranking of variables determining fog efficiency was essential for ensuring practical relevance and technical accuracy. While data driven techniques such as RF-RFE provide objective insights into feature importance, they may not fully capture the contextual, operational, or domain-specific factors influencing fog computing environments [99]. Industry experts bring in depth knowledge and real-world experience, allowing for critical assessment of whether the identified variables align with actual deployment challenges, application requirements, and system behavior under varying workloads.

Their validation helps bridge the gap between theoretical models and practical implementations, thereby enhancing the reliability and applicability of research findings. Moreover, expert feedback supports the refinement of variable selection, ensuring that any overlooked but practically significant parameters are considered. This collaborative approach aligns with best practices in applied computing, where interdisciplinary validation is encouraged to improve system performance and user relevance.

Studies emphasize the role of domain experts in enhancing the interpretability and trustworthiness of machine learning models used in IoT and fog systems [111], [109]. Therefore, integrating expert judgment is not only a methodological enhancement but also a critical step toward developing robust, efficient, and context-aware fog computing solutions.

By evaluating and ranking these efficiency variables, industry experts with real-world deployment expertise helped close the gap between academic models and actual applications. Their observations guaranteed that efficiency indicators correspond with real-world requirements, facilitating more accurate assessments and well informed system designs. In order to help academics and engineers optimize fog systems for

practical efficiency and effectiveness, this study uses industry experts to validate the ranking of fog efficiency determinant variables.

4.4.1 Data collection

A total of 30 industry experts were sampled as per the research design and a survey questionnaire in appendix 1 administered. The ten respondents who participated in the pilot study did not take part in the industry expert ranking. The survey results are discussed as follows.

4.4.2 Feedback from Respondents

There were a total of 30 respondents to the survey. There was gender balance among the respondents, with 50% indicating male and 40% female respectively. 10% of the respondents did not declare their gender as indicated in figure 4.4.

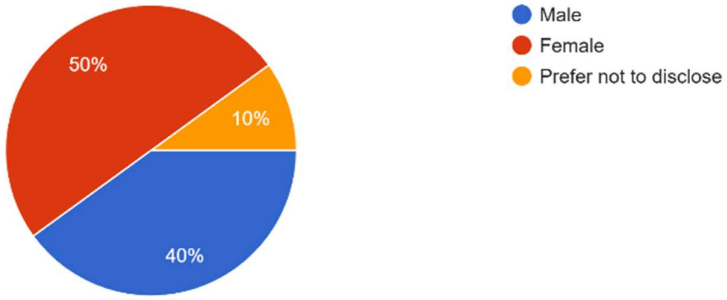


Figure 4.4: Distributions of respondents by gender.

The gender distribution results for the survey on ranking variables determining fog computing efficiency indicate balanced participation, enhancing the credibility and inclusiveness of the findings. A diverse respondent base ensures varied perspectives, which is essential in understanding complex technical issues like fog efficiency. If one gender is overrepresented, it may reflect existing trends in ICT engagement.

Majority of the respondents were fairly young, where 83.3% are between 18-24 years. 25.34% are between 25-34 years of age, as indicated in figure 4.2.

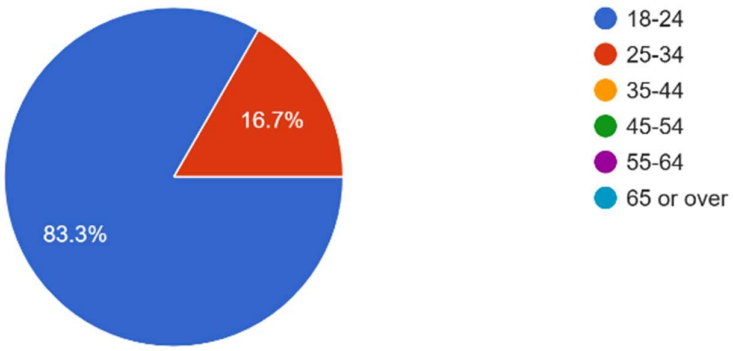


Figure 4.5 Distribution of Respondents by age

The results indicate that the majority of respondents (83.3%) were between 18–24 years, with an additional 25.34% aged 25–34 years. This suggests that the survey primarily attracted a younger demographic, likely reflecting the tech savvy nature of individuals engaged in IoT and fog computing. Their active involvement offers fresh insights.

All the respondents in this study already have knowledge of Fog Computing and have interacted with fog devices in their work or studies. This means they understand how fog systems operate, the role they play in processing data closer to IoT devices, and the challenges they face. Their prior experience ensures that the feedback and insights they provide are informed, practical, and directly relevant to the research objectives.

On the academic aspect, it was observed that 83.3% of the respondents have a bachelor's degree more to the professional training. 3.3% have a master's degree. 26.7% have other professional trainings more to the IOT related training. 50% have other post high school training which are not IT related.

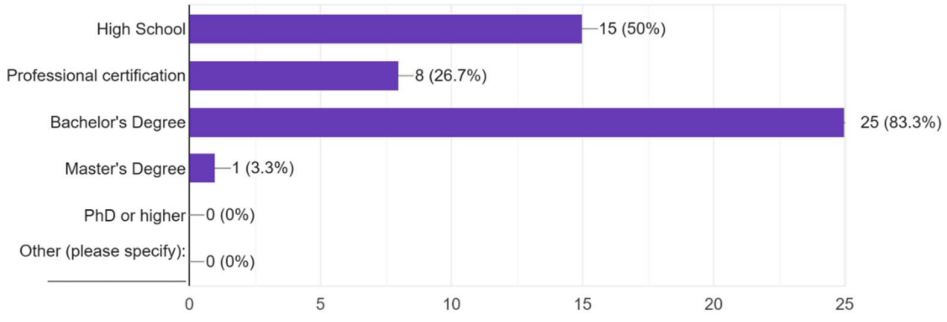


Figure 4.6: Academic levels of respondents.

This indicates a broad background of the respondents, suggesting that they have broad understanding of the subject.

On the role of fog computing in IOT, the majority (29.6%) said it is "fairly significant," while 18.5% deemed it "very significant" and another 25.9% labeled it "significant". (22.2%) considered it "extremely significant," and only 7.4% rated it not significant. With 92% of responses falling in the "significant" to "extremely significant" range it is a confirmation of the critical role played by fog computing in IOT, as presented in Figure 4.7.

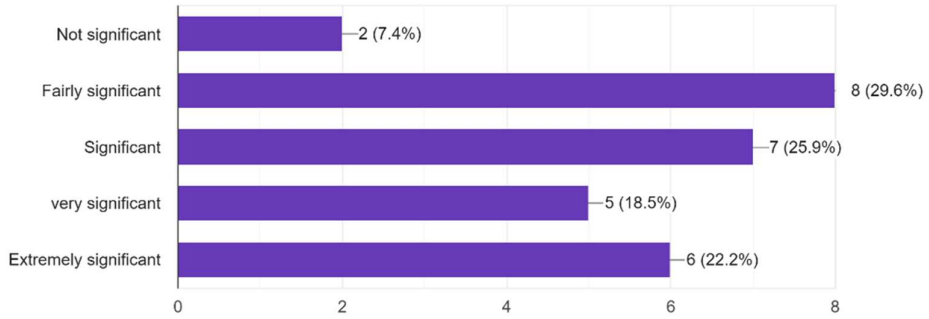


Figure 4.7: Rating role of fog computing in IOT

This affirms the critical role fog computing plays in addressing real-time data processing, latency, and scalability challenges, emphasizing the need for continued research and optimization in this area.

All the respondents have undergone professional training in IOT, majority of whom have gone through Cisco professional training as indicated in figure 4.8.

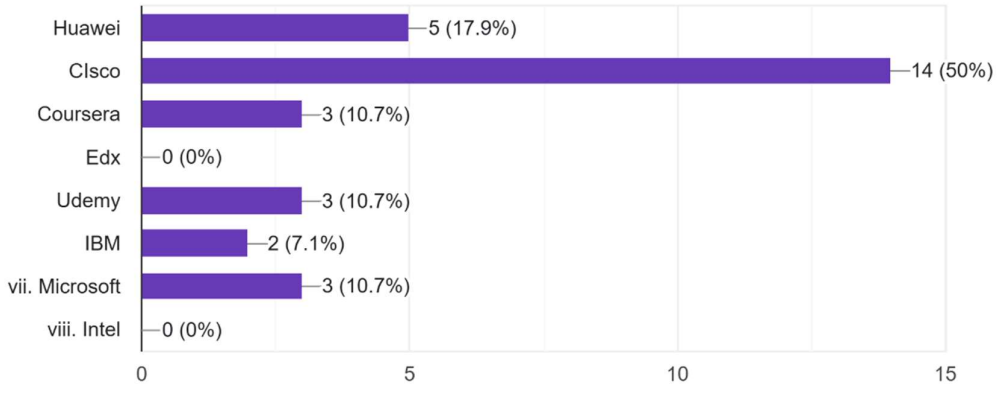


Figure 4.8: IOT Training of Respondents.

This highlights a well-informed participant base with credible technical expertise. Such training ensures that the responses are grounded in practical knowledge and

current industry standards, strengthening the reliability of insights gathered during the survey.

4.5 Analysis of Ranking Validation

Statistical analysis was carried out to evaluate the significance of the industry expert rating.

4.5.1 Kendall's Coefficient of Concordance

Kendall's Tau is a statistical metric used to evaluate the direction and strength of a link between two variables, especially when the data are ordinal or when parametric test assumptions like Pearson's correlation are not fulfilled. It is a non-parametric test that assesses the association based on the rankings of the data rather than the actual values, in contrast to Pearson's correlation, which requires interval or ratio data and assumes a linear relationship.

A value of +1 denotes a complete positive correlation, which means that as one variable rises, the other rises in a totally consistent way. The coefficient has a range of -1 to +1. A complete negative correlation, where one variable continually falls as the other rises, is represented by a value of -1. There is no systematic relationship between the ranks when the two variables have no correlation, as indicated by a value of 0.

Because it accounts for the influence of ties, Kendall's Tau is especially helpful when the data has a large number of repeated values. The coefficient is determined by counting the number of concordant and discordant pairs, where the rankings of the two variables are opposite in discordant pairs and aligned in concordant pairs (i.e., both variables grow or drop). In this regard, Kendall's Tau offers a trustworthy means of assessing association in situations when conventional correlation techniques may not be accurate because of assumptions being broken.

This measure is carried out to indicate the correlation among the variables in the ordinal data collected from industry experts. It assesses the degree of agreement between the ranked variables. Table 4.2 summarizes the ranking output. Other

analytical indicators have also been computed to give a more detailed outlook of the ranking.

Table 4.2: Responses from industry expert ranking

Respondents	Latency	Chanel Utilization	Throughput	Packet loss rate	Queue time
A	3	4	5	1	2
B	3	4	5	1	2
C	2	5	4	3	1
D	3	4	5	1	2
E	5	4	3	2	1
F	3	5	4	1	2
G	3	5	4	1	2
H	3	4	5	1	2
I	3	4	3	2	1
J	3	4	5	2	2
K	3	4	5	2	1
L	4	3	5	1	2
M	3	4	5	1	2
N	3	4	5	1	2
O	3	4	5	2	1
P	3	4	5	1	2
Q	3	5	4	1	2
R	4	5	3	2	1
S	3	4	5	1	2
T	3	5	4	2	1
U	3	4	5	2	1
V	3	4	5	1	2
W	1	3	5	3	2
X	3	1	5	3	2
Y	3	4	5	1	2
Z	3	5	4	2	1
AA	3	3	4	2	1
AB	3	4	5	1	2
AC	3	4	5	1	2
AD	3	5	4	1	2
Average	3.0	4.0	4.5	1.5	1.7

Kendall's value (W) of 0.728656361 indicates moderate to strong agreement among the respondents. They are largely in agreement, but there is still some variation in how they rank the fog computing efficiency variables. This suggests that the rankings are fairly consistent**,** but not perfectly aligned. The p-value was computed to be 0.0436568. Since the p-value is less than 0.05, it indicates statistically significant agreement at the 5% level.

This means that the observed level of agreement among the raters is unlikely to have occurred by random chance, and the rankings represent a real pattern of consensus. The Chi-square value is 84.52413793 for the ranking output. The relatively large Chi-square statistic supports the notion that the rankings show significant deviation from randomness. It indicates that the observed agreement is not due to chance, further confirming the statistical significance of the rankings.

Table 4.3: Descriptive results

Kendall's Value W:	0.728656361
Chi square:	84.52413793
P value:	0.0436568

The comparison between the RF-RFE ranking and the ranking given by industry experts shows that both sides strongly agree on which factors matter most for fog layer efficiency. Because a lower score means a higher level of importance, experts ranked packet loss rate (1.5) and queue time (1.7) as the most important factors, followed by latency (3.0), channel utilization (4.0), and throughput (4.5). The RF-RFE results followed the exact same pattern, placing packet loss rate first, queue time second, latency third, channel utilization fourth, and throughput fifth.

This close match shows that the machine learning model and the human experts identified the same key elements that influence performance. The small differences in the scores are negligible. This strong agreement suggests that the RF-RFE method reliably reflects expert opinion while adding the advantage of data-driven analysis. It also confirms that the developed hybrid DBSCAN–KNN model is well-aligned with practical industry understanding.

In simple terms, both the experts and the algorithm agree that reducing packet loss and managing queue time are the most effective ways to improve fog layer efficiency. This harmony between human insight and machine intelligence shows that the model is dependable and well-suited for enhancing performance in fog computing environments.

4.5.2 Significance of the Variables Ranking

The moderate to strong agreement (W=0.73) among the raters, along with the statistically significant p-value (0.0436568)**,** suggests that the rankings of the fog computing efficiency variables are reliable and meaningful. The raters largely agree on which variables are more important for fog computing performance. Given the statistical significance of the agreement, the rankings of the fog computing variables (Latency, Channel Utilization, Throughput, Packet Loss Rate, and Queue Time) can be considered trustworthy**.**

There is a strong pattern of consensus that the raters' rankings are not due to random chance. Since the ranking process shows statistical significance, these rankings can be used with more confidence in making decisions about the relative importance of the variables.

4.5.3 Correlation Analysis

The correlation analysis reveals key interdependencies among fog performance variables. This analysis is provided in table 4.3.

Table 4.3: Correlation analysis of the ranked variables

Variables	Latency	Throughput	Packet Loss Rate	Queue Time	Channel Utilization
Latency	1.00	-0.38	-0.29	-0.22	0.00
Throughput	-0.38	1.00	0.00	0.51	-0.41
Packet Loss Rate	-0.29	0.00	1.00	-0.59	0.00
Queue Time	-0.22	0.51	-0.59	1.00	0.00
Channel Utilization	0.00	-0.41	0.00	0.00	1.00

Latency negatively correlates with throughput (−0.38), packet loss rate (−0.29), and queue time (−0.22), suggesting performance may decline with higher delays. Channel utilization also negatively affects throughput (−0.41) and other metrics, indicating reduced efficiency under heavy loads. Throughput and queue time show a moderate positive correlation (0.51), while packet loss and queue time are strongly negatively correlated (−0.59), emphasizing the role of buffering in performance optimization.

The correlation analysis highlights significant interdependencies among fog performance variables, offering valuable insights into system behavior. Higher latency and channel utilization are linked to reduced throughput and efficiency, indicating the need for careful load management. The strong negative correlation between packet loss and queue time underscores the importance of effective buffering strategies. These findings emphasize the need to optimize delay-sensitive parameters and resource allocation to enhance overall fog performance in real-world applications.

Figure 4.9 compares the variable rankings from the RF-RFE method and expert evaluations. Lower ranks indicate higher importance.

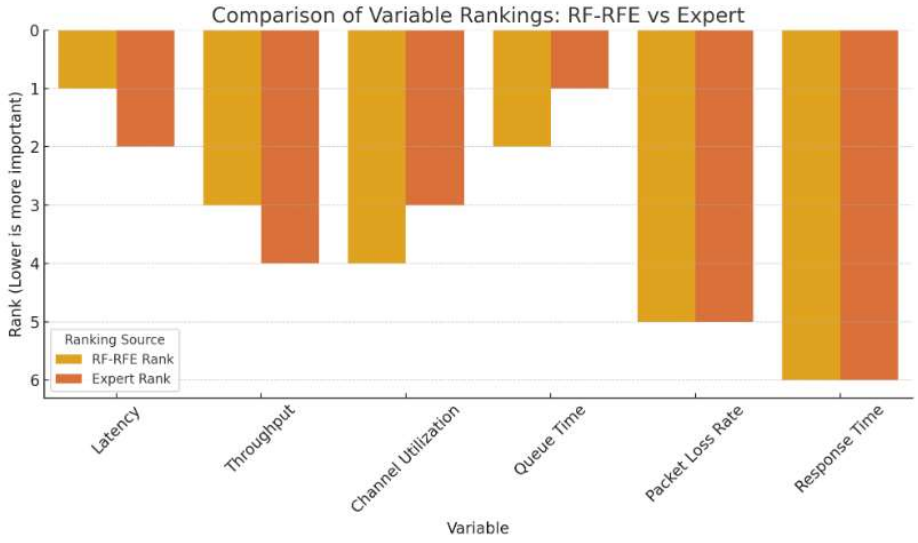


Figure 4.9: Comparison of variable ranking between RF-RFE and experts

The figure compares variable rankings determined by the RF-RFE algorithm and industry experts. Both methods agree on the high importance of packet loss rate and response time. Minor variations appear in ranking latency and queue time, suggesting the algorithm's alignment with expert insights.

4.6 Discussion

By focusing on the most important performance indicators, researchers can simplify their analysis and find more effective ways to improve fog computing systems. According to the ranking results, packet loss rate comes out on top because it directly impacts how reliably data is delivered. When too many packets are lost, the system

needs to resend them, which slows everything down and reduces performance making it a key issue to fix.

Queue time ranks second since it shows how long data has to wait before being processed. Shorter queues mean faster responses and better use of computing resources. Latency, which comes in third, measures how long it takes for data to travel across the network. In fog computing, especially for real time IoT applications, keeping latency low is crucial.

Channel utilization, ranked fourth, looks at how efficiently bandwidth is being used. It's still important, but its impact often depends on the other factors. Finally, throughput the rate at which data is transmitted is fifth. While higher throughput is good, it can be heavily affected by packet loss and latency. This ranking helps guide efforts to improve performance in a balanced, effective way.

Response time was identified as a variable determining fog efficiency due to its relevance in evaluating the speed at which a system handles tasks. However, it was eliminated from the final set of variables by the RF-RFE process. After detailed analysis, it was revealed that it had overlapping contribution with other critical variables. Including response time as a separate variable would introduce redundancy and multi-linearity into the model, violating the assumption of independence among variables.

To validate this ranking, industry experts were engaged, offering professional insight and grounding the findings in real-world relevance. The moderate to strong agreement (Kendall's W = 0.7287, $p < 0.05$) confirms a reliable consensus on variable importance. Demographic data showed a youthful, technically trained group, with many Cisco-certified professionals, further strengthening the credibility of the responses. A significance rating of over 92% for fog computing's role in IoT reaffirms is critical value.

Correlation analysis added depth, revealing how latency and packet loss negatively influence throughput and queue time, while buffering appears to mitigate packet loss. These insights collectively support the accuracy of the variable ranking and

demonstrate the need for targeted optimization in fog environments. The study's methodology, validation, and analysis offer a robust foundation for future model development and highlight the necessity of precise variable identification in enhancing fog performance for IoT systems.

The comparison between RF-RFE and expert rankings demonstrates strong alignment in identifying key variables such as packet loss rate and response time, which are critical to fog efficiency. According to [122] machine learning-based feature ranking methods like RF-RFE provide robust, data-driven insights that often mirror expert intuition. Similarly, [104] emphasize the reliability of combining algorithmic and expert evaluations to validate feature importance in IoT systems.

The top ranked variables packet loss rate, response time, and latency align closely with the conceptual framework grounded in the Quality of Service theory, which emphasizes efficiency, reliability, and responsiveness in fog computing. These variables directly reflect the performance dimensions critical to service delivery in the fog layer. Their prioritization confirms the theoretical assumptions that minimizing delays and transmission errors significantly enhances overall IoT system performance and user experience in decentralized environments.

The correlation analysis shows how different factors in the fog layer affect each other. When latency, increases, throughput, packet loss, and queue time all drop, meaning performance gets worse with longer delays. High channel utilization also reduces throughput, showing that heavy traffic can slow the system down. On the other hand, throughput and queue time rise together, which suggests that giving data more time in the queue can improve flow. Packet loss and queue time are strongly linked in the opposite way, with longer queues helping reduce data loss. These patterns are key for improving fog efficiency.

4.7 Chapter Summary

This chapter covers a systematic literature review, which investigates the variables determining efficiency of fog layer in the Internet of Things with Justification as to why the variables are considered the key determinants. RF-RFE is used to rank the

identified variables using a synthetic data set, which is generated using simpy python framework. Once the variables are ranked, industry experts are used to validate the ranking.

CHAPTER FIVE
DESIGNING A DBSCAN AND KNN BASED HYBRID MODEL

5.1 Introduction

The title reflects an effort to combine the strengths of two popular machine learning techniques to create a more effective solution for complex data environments. DBSCAN is known for its ability to detect clusters of varying shapes and handle noise, while KNN excels at classifying data points based on similarity. However, each method has its limitations when used alone DBSCAN can struggle with varying densities, and KNN can be sensitive to noisy or irrelevant data.

By designing an algorithm that blends these two algorithms, the goal is to overcome their individual weaknesses and build a more adaptive, accurate, and efficient model. This hybrid design is especially useful in fog computing, where data is often dynamic and context dependent. The combined approach promises better performance in detecting patterns, improving accuracy, and supporting real time decision making in complex computing environments.

5.2 The DBSCAN Algorithm

Density Based Spatial Clustering of Applications with Noise (DBSCAN) is an unsupervised clustering algorithm that identifies clusters of varying shapes and sizes in a dataset based on the concept of density. Unlike K-Means, DBSCAN does not require specifying the number of clusters beforehand. Instead, it groups points that are densely packed and marks points in sparse regions as noise (outliers). It is particularly effective for arbitrary-shaped clusters and outlier detection, making it ideal for applications such as geospatial analysis, anomaly detection, and IoT fog computing.

BSCAN operates based on two important parameters: (Epsilon radius) which defines the neighborhood radius around a data point. MinPts (Minimum Points): The minimum number of points required to form a dense region. DBSCAN classifies each point into one of the following: Core Point: A point with at least minPts neighbors within ε radius. Border Point: A point within ε radius of a core point but with fewer

than minPts neighbors. Noise (Outlier): A point that is neither a core nor a border point.

The pseudocode for DBSCAN is as follows:

```
FUNCTION DBSCAN (dataset, eps, minPts):
  # Initialize
   Labels = [UNVISITED] * Len (dataset) # Track visited/noise/cluster assignments
   Cluster id = 0
      FOR EACH point IN dataset:
      IF labels [point]! = UNVISITED:
         CONTINUE
# Skip already processed points
# Get all points within epsilon distance
      Neighbors = find neighbors (point, dataset, eps)
            IF Len (neighbors) < min pts:
         Labels [point] = NOISE
# Mark as noise
         CONTINUE
# Start new cluster
      Cluster id += 1
      Labels [point] = cluster id
# expand cluster
      Queue = neighbors copy ()
      WHILE queue not empty:
         Current pt = queue. Pop (0)
         IF labels [current pt] == NOISE:
            Labels [current pt] = cluster id # Change noise to border point
         IF labels [current pt]! = UNVISITED:
            CONTINUE # Skip processed points
                  Labels [current pt] = cluster
 # Find current point's neighbors
         Current neighbors = find neighbors (current pt, dataset, eps)
               IF Len (current neighbors) >= min pts:
            Queue. Extend (current neighbors) # Expand cluster further
```

```
        RETURN labels
# Helper function to find neighbors
FUNCTION find neighbors (point, dataset, and eps):
    Neighbors = []
    FOR EACH other pt IN dataset:
        IF Euclidean distance (point, other pt) <= eps:
            Neighbors. Append (other pt)
    RETURN neighbors.
```

The pseudo code implements the DBSCAN clustering algorithm, which groups data points based on density. It begins by marking all points as unvisited. For each unvisited point, it identifies neighboring points within a specified radius (eps). If the number of neighbors is less than a minimum threshold (minpts), the point is marked as noise. Otherwise, a new cluster is formed, and the point is added to it. The algorithm then expands the cluster by examining each neighbor. If neighbors have sufficient nearby points, they are also added to the cluster. This process continues until all reachable points are processed.

5.3 The KNN Algorithm

The K-Nearest Neighbors (KNN) algorithm is a supervised learning algorithm used for classification and regression. It is based on the idea that similar data points exist close to each other in feature space. KNN is a lazy learning algorithm, meaning it does not learn a model during training but makes predictions by finding the most similar (nearest) training samples. In classification, it assigns a class label based on the majority vote of nearest neighbors. In regression, it predicts a continuous value by averaging the values of nearest neighbors.

The pseudocode for the KNN algorithm is as follows:

```
FUNCTION KNN(X train, y train, x new, k):
    # Inputs:
    #   X train: Training feature vectors
    #   y train: Corresponding labels
    #   x new: New data point to classify
    #   k: Number of neighbors to consider
    # Step 1: Calculate distances between x new and all training points
```

```
    Distances = []
    FOR i FROM 0 TO LENGTH(X train) - 1:
        Dist = EUCLIDEAN_DISTANCE (x new, X train [i])
        Distances. Append ((dist, y train [i])) # Store (distance, label)
# Step 2: Sort distances and pick top-k neighbors
    SORT (distances) # Ascending order by distance
    K nearest = distances [0: k] # Get first k entries
# Step 3: Majority vote for classification
    Label counts = {}
# Dictionary to count labels
    FOR (dist, label) IN k nearest:
        IF label IN label counts:
            Label counts [label] += 1
        ELSE:
            Label counts [label] = 1
# Step 4: Return the most frequent label
    Predicted label = ARGMAX (label counts) # Label with highest count
    RETURN predicted label
# Helper function for Euclidean distance
FUNCTION EUCLIDEAN_DISTANCE (a, b):
    Sum sq = 0
    FOR i FROM 0 TO LENGTH (a) - 1:
        Sum sq += (a[i] - b[i]) ** 2
    RETURN SQRT (sum sq)
```

The pseudocode outlines the K-Nearest Neighbors (KNN) classification algorithm. It takes in training data (X_train and y_train), a new data point (x_new), and a number k to determine how many neighbors to consider. First, it computes the Euclidean distance between x_new and each training sample, storing these distances along with the corresponding labels. The distances are then sorted in ascending order, and the k closest neighbors are selected. A majority vote among these neighbors determines the predicted label for x_new. The label with the highest count is returned as the final classification output.

5.4 The Hybrid Algorithm

The Hybrid DBSCAN+KNN Model combines the strengths of Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and K-Nearest Neighbors (KNN) to enhance efficiency in tasks for clustering, classification, and anomaly detection. DBSCAN is effective in finding arbitrarily shaped clusters and detecting noise. However, it does not classify new data points once clustering is complete.

KNN algorithm assigns labels to new points based on their similarity to known data. By combining DBSCAN and KNN, we create a hybrid model that first clusters data using DBSCAN and then classifies new points using KNN, improving accuracy, noise handling, and scalability.

5.4.1 Pseudocode for Hybrid the Algorithm

The following pseudo code expresses the stages involved in execution of the model.

```
# HYBRID DBSCAN+KNN MODEL FOR FOG OPTIMIZATION
FUNCTION optimize fog efficiency (fog nodes data):
   # Input: fog nodes data = [packet loss, queue time, latency, channel util, throughput]
   # 1. Normalize data (scale all metrics to 0-1 range)
   Normalized data = Standard Scaler ().fit transform (fog nodes data)
   # 2. DBSCAN for anomaly detection
   Dbscan = DBSCAN (eps=0.5, min samples=5)
   Clusters = dbscan. Fit predict (normalized data)
   # 3. Label data: -1=noise (bad nodes), 0=normal, 1=optimal
   Labels = [0 if x == -1 else 1 for x in clusters] # Simple binary labeling
   # 4. Train KNN only on non-noise data
   Clean data = normalized data [clusters! = -1]
   Clean labels = labels [clusters! = -1]
   Knn = K Neighbors Classifier (n neighbors = 3)
   Knn.fit (clean data, clean labels)
   # 5. Predict all nodes (including former noise)
   Predictions = knn. Predict (normalized data)
    # 6. Efficiency optimization actions
   FOR each node IN fog nodes:
```

```
IF predictions [node] == 0:  # Bad performer
   # Apply corrective measures
   Node. Packet loss = apply packet loss fix (node)
   Node. Queue time = redistribute queue (node)
   Node. Latency = reroute traffic (node)
   Node. Channel util = balance channel load (node)
   Node. Throughput = adjust bandwidth (node)
RETURN optimized metrics.
```

The hybridization method is a sequential process in which KNN classifies new data points based on the structured dataset after DBSCAN first refines the dataset by clustering and eliminating noise. For both models, data pretreatment guarantees that the data is appropriately cleaned, standardized, and formatted. Separate training and validation of each model aids in efficiency optimization, while cross validation and hyper parameter adjustment improve accuracy.

Following training, weighted averaging is used to aggregate the models' outputs, allocating various weights according to how well each model performs. Lastly, metrics including accuracy, precision, and recall are used to compare the hybrid model against individual models. The hybrid approach boosts decision making, lowers noise, and improves accuracy in real-world applications like anomaly detection and IoT fog computing by combining KNN's classification with DBSCAN's density based clustering.

5.4.2 Flow Chart for the Hybrid Model

The flowchart outlines a structured approach to clustering and classification using two key machine learning algorithms, DBSCAN and KNN**.** This workflow is designed to group data points into clusters based on density and then classify them using a supervised learning method. The process concludes efficiency tuning to optimize the results. The algorithm function in illustrated in the flowchart in figure 5.1.

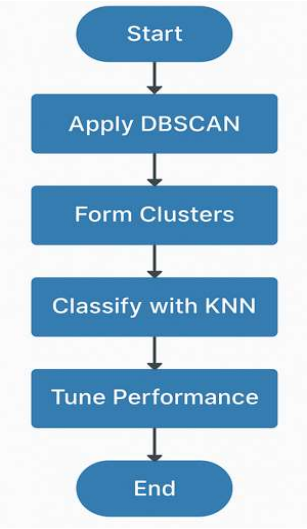


Figure 5.1: Flow chart for the hybrid Algorithm

The step-by-step approach ensures robust clustering and classification**,** making it ideal for applications where data is noisy, unstructured, or requires both unsupervised clustering and supervised learning**.**

5.5 Mathematical Expression of the Hybrid Algorithm

The comprehensive mathematical formulation of this hybrid model is as follows:

Let:

$D = \{x_1, x_2, ..., x_n\} \subseteq \mathbb{R}^d$ be the dataset collected from IoT sensors in the fog environment, where each $x_i \in \mathbb{R}^d$ represents a d-dimensional feature vector.

Step 1. Apply DBSCAN to detect and eliminate noise:

$DBSCAN(D, \varepsilon, MinPts) \rightarrow D_clean \cup D_noise$

Where:

ε is the neighborhood radius

MinPts is the minimum number of points to form a dense cluster

$D_clean \subseteq D$ are the core and border points (non-noise)

$D_noise = D \setminus D_clean$ are the outliers identified by DBSCAN

Step 2. Apply KNN to classify the cleaned dataset:

$f_KNN : D_clean \rightarrow C$

$f_KNN(x_i) = argmax_l \sum_\{j \in N_k(x_i)\} I(y_j = l)$,

Where:

$N_k(x_i)$ is the set of k-nearest neighbors of x_i

y_j is the label of neighbor j

I(·) is the indicator function returning 1 if the condition is true, otherwise 0

Step 3. Hybrid Model Function

$H(x_i) = f_KNN(x_i), \forall\ x_i \in D_clean = DBSCAN_Filter(D, \varepsilon, MinPts)$

Thus, the overall hybrid model H can be expressed as:

$H : D \rightarrow C \cup \{null\}$,

$H(x_i) = \{ f_KNN(x_i), \text{if } x_i \in D_clean; null, \text{if } x_i \in D_noise \}$

This model enhances performance in the fog layer by: Reducing noisy data before classification (improving accuracy), Lowering processing overhead (increased efficiency), Providing robust classification in heterogeneous IoT environments

The mathematical expression defines a hybrid model combining DBSCAN and KNN to enhance classification accuracy and efficiency in fog computing environments. A dataset D={x1,x2,...,xn}⊆RdD = \{x_1, x_2, ..., x_n\} \subseteq \mathbb{R}^dD={x1,x2,...,xn}⊆Rd represents IoT sensor data, where each point is a d-dimensional vector. DBSCAN is first applied to partition the dataset into clean data DcleanD_{\text{clean}}Dclean and noise DnoiseD_{\text{noise}}Dnoise using neighborhood radius ε\varepsilonε and a minimum point threshold MinPts\text{MinPts}MinPts.

The cleaned data is then classified using KNN, where each data point is assigned the label most common among its kkk-nearest neighbors. The final hybrid model H(xi)H(x_i)H(xi) classifies only noise free data while ignoring outliers. This two-step process improves performance by filtering noise before classification, thus enhancing accuracy, reducing computation, and ensuring robustness in heterogeneous IoT environments typically found in fog computing architectures.

5.6 Sequence Diagram for the Hybrid Model

The sequence diagram in figure 5.2 illustrates the hybrid operation of DBSCAN and KNN algorithms within a fog computing environment to enhance data processing.

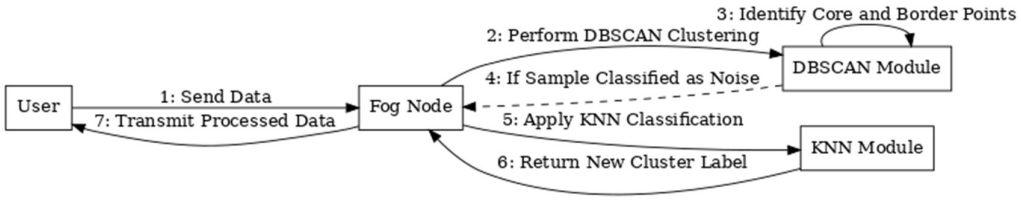


Figure 5.2: Sequence diagram for the Hybrid operation

The diagram illustrates the operation of a hybrid model that integrates DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and KNN (K-Nearest Neighbors) to enhance fog computing efficiency. The process begins with the User (sensor)**,** sending raw data to the Fog Node (Step 1). The Fog Node initiates DBSCAN clustering (Step 2), which groups data points based on density.

During this stage, core and border points are identified to determine which data points belong to well-defined clusters (Step 3). If DBSCAN detects any data points as noise meaning they do not fit into any cluster (Step 4) these points are forwarded for further analysis.

The KNN Module then performs classification on these noisy samples by comparing them to their nearest neighbors (Step 5). Once a suitable label or cluster assignment is determined, the KNN Module sends the new cluster label back to the Fog Node (Step 6). Finally, the Fog Node compiles all processed data, including DBSCAN clusters and KNN labeled noise, and transmits the final results back to the User (Step 7). This hybrid approach ensures accurate clustering in dynamic fog environments by combining DBSCAN's ability to discover clusters with KNN's ability to handle outliers.

5.7 Discussion

The integration of DBSCAN and KNN into a hybrid algorithm offers a strategic and technically sound approach to addressing the complex challenges of fog computing environments, especially where data is voluminous, dynamic, and noisy. As established in the previous chapter, determining and ranking fog efficiency variables provided a foundational step for system optimization. Building upon this, the current discussion introduces a robust model that leverages the strengths of two machine

learning algorithms DBSCAN and KNN to improve fog layer performance in Internet of Things (IoT) systems.

Technically, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) serves as the first phase in the hybrid architecture. It is an unsupervised learning algorithm that relies on two primary parameters: epsilon (ε) the radius for neighborhood consideration and minPts the minimum number of points needed to form a dense cluster. According to the DBSCAN pseudocode, the algorithm iterates over each data point, finding neighbors within ε distance.

If a point has fewer than minPts neighbors, it is labeled as noise. Otherwise, a new cluster ID is assigned, and the cluster expands recursively through its neighbors. The find neighbors () helper function, which uses Euclidean distance, is central to identifying the local density around points. This process ensures that DBSCAN effectively isolates meaningful clusters while filtering out noise, making it particularly effective for spatial data or irregular cluster shapes common in IoT fog systems.

However, DBSCAN alone is inadequate for classifying new, unseen data this is where KNN is employed in the second phase. As a supervised learning algorithm, KNN classifies a data point by finding the majority class among its k nearest neighbors. The KNN pseudocode illustrates the distance calculation between the input point and all training data, sorting these distances, and selecting the top k closest samples. A majority voting mechanism is then used to assign the final label. The advantage of this approach lies in its adaptability to newly incoming fog data points, which may not have been present during initial clustering.

The hybrid algorithm's workflow depicted in both the flowchart and sequence diagram begins with data normalization, followed by DBSCAN clustering. Clusters and noise samples are separated, and the non-noise samples are used to train the KNN model. Subsequently, KNN classifies all data, including those initially marked as noise. The sequence diagram illustrates this interaction where user data is sent to the fog node, processed through DBSCAN, and then re-evaluated by KNN if initially flagged as noise. Finally, processed data, now labeled and refined, is transmitted back, improving system accuracy and responsiveness.

Mathematically, the hybrid model is expressed as:

H(xi)=fKNN(xi)H(x_i) = f_{KNN}(x_i)H(xi)=fKNN(xi)

if xi∈Dcleanx_i \in D_{clean}xi∈Dclean,

and H(xi)=nullH(x_i) = \text{null}H(xi)=null

Otherwise, where DBSCAN acts as a noise filter and KNN as a label assigner. This formulation reduces classification errors, handles anomalies, and enhances processing speed key requirements for real-time fog computing. The resulting system is resilient, scalable, and well suited for diverse IoT scenarios.

5.8 Chapter Summary

This chapter presented a hybrid machine learning architecture integrating Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and k-Nearest Neighbors (KNN). The goal is to optimize data processing within resource-constrained, dynamic, and noise-heavy IoT fog computing layers. Building upon the fog efficiency variables identified in previous analysis. This two-phase model balances robust noise filtration with adaptable classification capabilities for real-time edge environments.

CHAPTER SIX
DEVELOPMENT OF HYBRID MODEL TO IMPROVE FOG EFFICIENCY

6.1 Introduction

This chapter adopts a Design Science approach to develop a hybrid model that enhances fog performance by integrating DBSCAN and KNN algorithms. Through iterative design, implementation, and evaluation, the model is constructed as a purposeful artifact to address noise filtering and accurate classification in fog environments. Design Science ensures that the hybrid solution is both theoretically grounded and practically effective for real-world fog computing challenges.

6.2 Development Process of the Hybrid Model

Design Science Research and sequence diagram support clear communication and systematic development. Both are discussed below:

6.2.1 Design Science Approach

Design Science Approach provides a structured approach for developing innovative solutions to real-world problems, making it ideal for designing a hybrid model that combines DBSCAN and KNN to improve fog efficiency. It guides the research through cycles of problem identification, solution building, and evaluation. To visually represent how the hybrid model works step by step, a sequence diagram is used.

The diagram shows how DBSCAN and KNN interact with system components during data processing, making it easier to understand the model's logic and flow. Together, DSR and the sequence diagram support clear communication and systematic development. Figure 6.1 shows the design science research process which culminates in artifact development.

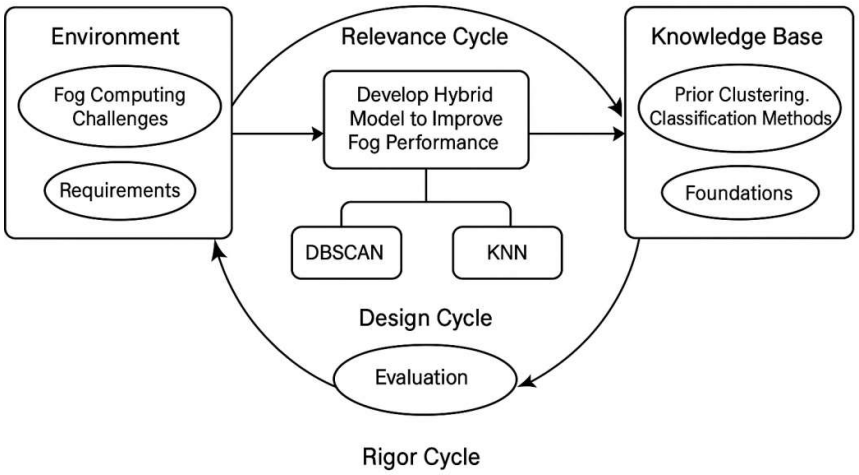


Figure 6.1: Design Science Process

The diagram illustrates the Design Science process used to develop a hybrid model that combines DBSCAN and KNN to improve fog computing performance. It shows three key parts: the environment, which includes fog computing challenges and requirements; the knowledge base, which provides existing clustering and classification methods; and the central design cycle, where the hybrid model is developed and evaluated. The relevance cycle connects the real-world needs to the model's design, while the rigor cycle ensures the model is grounded in existing knowledge. This process ensures the solution is practical, evidence based, and tailored to real fog computing challenges.

To implement the design science research outlined in the diagram, a structured and iterative approach was followed across three interconnected cycles: relevance, design, and rigor. The process began with the Relevance Cycle, where real-world fog computing challenges such as latency, packet loss rate, throughput, Channel utilization and queue time were identified through literature review and expert review. These insights informed the design cycle, which focused on developing a hybrid model combining DBSCAN and KNN algorithms.

DBSCAN was used for clustering noisy and irregular IoT data, while KNN provided classification based on proximity, enhancing decision-making at the fog layer. The model was iteratively refined through simulation and feedback. In parallel, the Rigor Cycle ensured the model was grounded in existing theoretical and empirical

knowledge, drawing from prior clustering and classification research. The hybrid model was then tested using performance metrics such as precision, recall and Support. Benchmark comparisons validated the model's effectiveness.

6.2.2 Sequence Diagram for the Model Development

Developing the hybrid model involved combining two algorithms to leverage their strengths. Figure 6.2 shows the stages involved in implementing the model. This model presents an innovative approach to enhance fog computing efficiency by combining DBSCAN for clustering and KNN for classification.

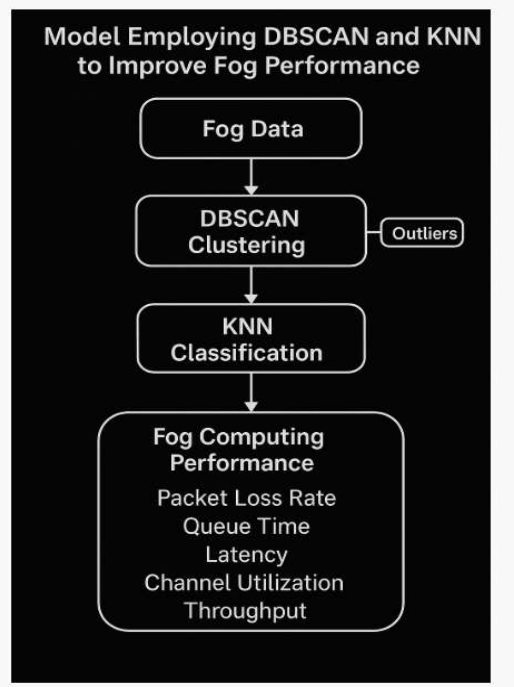


Figure 6.3: Stages of the model development

The process begins with fog dataset collection, a set which includes critical variables: packet loss rate, queue time, latency, channel utilization, and throughput. The DBSCAN algorithm was then applied to cluster this data, effectively grouping similar data points while identifying and isolating outliers, an essential step for detecting anomalies or irregular patterns in network behavior.

These refined clusters are subsequently fed into a KNN classifier, which leverages proximity based learning to categorize and predict efficiency outcomes with greater accuracy. By integrating unsupervised clustering (DBSCAN) with supervised classification (KNN), the model achieves a more robust analysis of fog computing

environments. This hybrid approach not only improves the detection of efficiency bottlenecks but also optimizes resource allocation and decision-making in real time.

The result is a significant enhancement across key fog efficiency metrics, ensuring efficient, low-latency, and reliable operations in the fog layer. This model is particularly valuable for IoT and edge computing applications, where data variability and dynamic conditions demand adaptive and precise analytical tools.

6.3 DBSCAN Performance

DBSCAN is an unsupervised clustering algorithm that groups data based on density without requiring a preset number of clusters. It uses two key parameters: eps (radius) and minPts (minimum points for a cluster). Points are classified as core, border, or noise. The process involves data normalization, parameter tuning using k-distance graphs, and clustering. Efficiency is evaluated using metrics like V-Measure, ARI, and Silhouette Score by comparing results to ground truth labels, helping refine the model

Synthetic data for evaluating the DBSCAN algorithm was generated by modeling variables, which are packet loss, latency, queue time, channel utilization, and throughput to reflect real network behavior. Variables are simulated using normal distributions with added noise to mimic real world fluctuations. The data undergoes standardization before DBSCAN is applied to identify clusters and anomalies. The evaluation process involves generating the dataset, applying DBSCAN, and assessing clustering performance to detect patterns and potential signs of degraded fog computing network performance. The results are as shown in the figure 6.3.

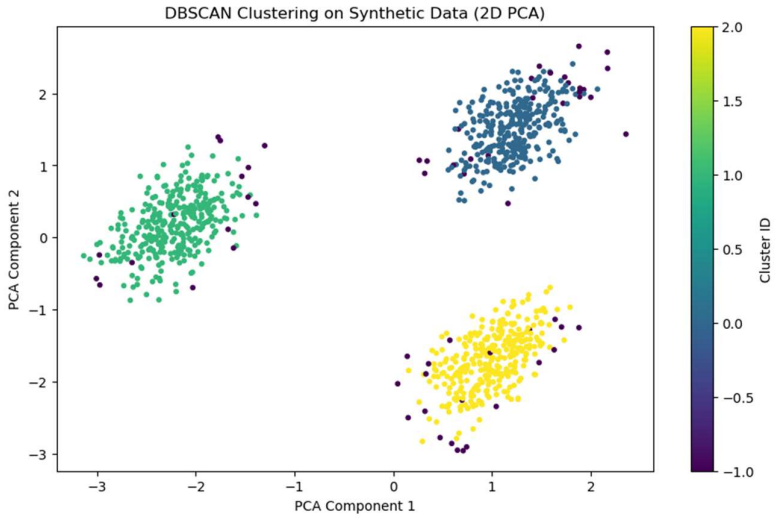


Figure 6.4 DBSCAN clustering.

Number of clusters found: 3
Number of noise points: 61
Silhouette Score: 0.5962.

The dataset was subjected to the DBSCAN clustering algorithm, which produced intriguing outcomes. Based on these network efficiency indicators, a total of three clusters were found, suggesting that the data naturally forms three different groupings. The efficiency of the network is likely to exhibit distinct patterns or behaviors, and these clusters may correlate to different network situations, including high, medium, and low performance. Nevertheless, 61 data points were also classified as noise by the algorithm, indicating that they did not fit well into any of the three clusters.

The Silhouette Score for cluster quality was 0.5962. This score indicates how closely each point resembles its own cluster in relation to other clusters. Since numbers nearer 1 denote a strong clustering structure while scores nearer -1 denote weak separation between clusters, a score of 0.5962 indicates that the grouping is reasonably well-defined. Although not flawless, this score suggests that the three clusters that DBSCAN created are significant and that the points that make up each cluster are reasonably coherent. Overall, the findings imply that DBSCAN was successful in identifying underlying patterns in the network's performance.

6.4 Experimental Evaluation of KNN

The evaluation is based on the code displayed in appendix 5. The following steps were involved: Generation of synthetic data, Generation of labels (1: good performance, 0: poor performance), Converting to a DataFrame, Splitting into features and target, Splitting into training and testing sets, Initializing and train the KNN classifier, Prediction and evaluation of classifier, Generation of evaluation metrics and lastly Generation of Confusion matrix. This process was repeated for evaluation of all the identified variables.

In fog computing models, efficiency is measured using several key metrics. Precision evaluates the accuracy of positive predictions, while recall assesses the model's ability to identify all relevant positive instances. The F1-score balances precision and recall, offering a single metric that accounts for both false positives and false negatives. Accuracy measures the proportion of correct predictions overall. The macro average calculates the average efficiency across all classes, treating them equally, whereas the weighted average adjusts for class imbalances by weighting efficiency based on class frequency. These metrics provided a comprehensive evaluation of model performance. Figure 6.3 shows a confusion matrix for testing KNN on the generated dataset to create baseline values with which to compare other outputs.

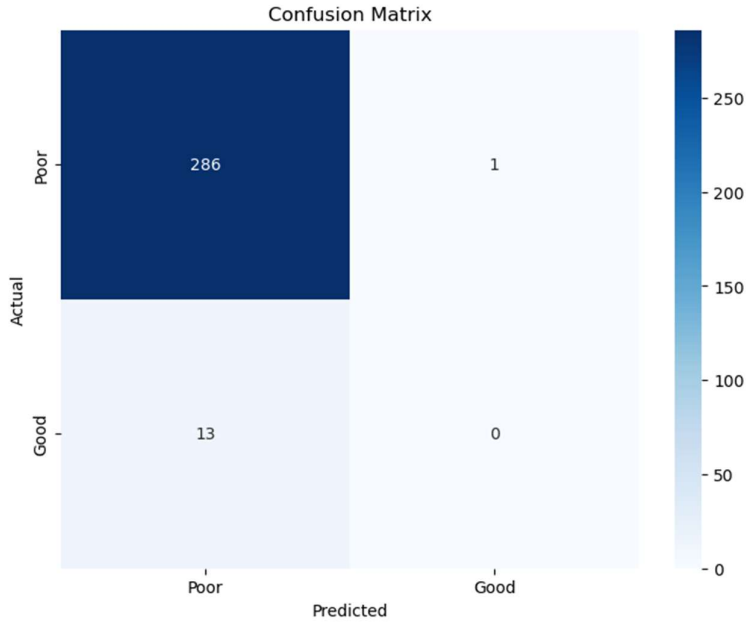


Figure 6.5: Confusion matrix results for KNN.

The confusion matrix in figure 6.4 shows the performance of a binary classifier distinguishing between “Poor” and “Good” fog performance. The model correctly identified 286 “Poor” instances but misclassified 13 “Good” cases as “Poor.” Only one “Poor” case was incorrectly predicted as “Good,” and no “Good” cases were correctly classified. This indicates high sensitivity to detecting “Poor” conditions but poor precision for the “Good” class, suggesting an imbalanced dataset or model bias towards the “Poor” category. Table 6.1 shows a baseline performance of KNN algorithm based on the five identified variables.

Table 6.1: Experimental output of KNN on All Variables

Class	Precision	recall	F1-score	Support
0	0.98	0.97	0.98	287
1	0	0	0	13
Accuracy			0.9367	300
Macro avg	0.48	0.49	0.48	300
weighted avg	0.91	0.94	0.93	300

The K-Nearest Neighbors (KNN) algorithm's results demonstrate notable efficiency differences between the two classes while demonstrating great overall accuracy. Overall, the model performs well with an accuracy of 93.67%. Nevertheless, Class 1's accuracy, recall, and F1-score are all zero, suggesting that the model is unable to accurately classify instances of this class.

With precision (0.98), recall (0.97), and an F1-score (0.98) for Class 0, the model performs very well, proving that KNN accurately classifies the majority of these cases. While the weighted average F1-score (0.93) stays high because Class 0 dominates the sample, the macro-average F1-score (0.48) shows the efficiency gap between the classes.

6.5 Experiment on the Hybrid Model

Some specific parameters are used to evaluate efficiency of the model. They include: Precision, which is the ratio of correctly predicted positive observations to the total predicted positive observations (true positives + false positives). The Formula is Precision = TP / (TP + FP). Precision tells us how many of the predicted positives are actually positive. Recall (Sensitivity or True Positive Rate) is the ratio of correctly

predicted positive observations to all actual positive observations (true positives + false negatives). Formula: Recall = TP / (TP + FN). Recall tells us how many actual positives are captured by the model.

F1-Score: is the harmonic mean of Precision and Recall. It balances both Precision and Recall in a single metric. The F1-score is useful when the class distribution is imbalanced. Formula: F1 = 2 * (Precision * Recall) / (Precision + Recall). It helps evaluate the model when you need to balance false positives and false negatives. Support: refers to the number of actual occurrences of the class in the dataset. It indicates how many instances belong to each class in the dataset.

Accuracy: is the ratio of correctly predicted observations to the total observations. Formula: Accuracy = (TP + TN) / (TP + TN + FP + FN). It measures how often the classifier is correct overall. Macro Average: computes the metrics (Precision, Recall, and F1-Score) for each class independently and then takes the average, treating all classes equally. Macro Avg = (Precision for Class 1 + Precision for Class 2 + ... + Precision for Class N) / N.

Weighted Average: computes the metrics (Precision, Recall, and F1-Score) for each class, but it gives more weight to the classes with more instances (higher support). Formula: Weighted Avg = (Support of Class 1 * Metric of Class 1 + Support of Class 2 * Metric of Class 2 + ... + Support of Class N * Metric of Class N) / Total Support. It ensures that larger classes contribute more to the final average.

The code in appendix 6 was created to generate a hybrid model combining DBSCAN (Density-Based Spatial Clustering of Applications with Noise) for preprocessing and KNN for classification. This model aims to improve fog layer efficiency prediction using synthetic data based on Packet Loss Rate, Queue Time, Latency, Channel Utilization, and Throughput. DBSCAN first clusters the data to detect noise and meaningful groups, helping to improve KNN's efficiency on a refined dataset. The test results are displayed in table 6.2.

Table 6.2: Output of Hybrid Model on all variables

Hybrid Model				
	Precision	**recall**	**F1-score**	**Support**
0	0.96	1	0.98	287
1	0	0	0	13
Accuracy			0.9533	300
Macro Avg	0.48	0.5	0.49	300
weighted avg	0.92	0.95	0.93	300

This comparative analysis demonstrates the clear superiority of the Hybrid DBSCAN+KNN model over the standard KNN Baseline for this specific classification task, particularly concerning the most critical category: Class 0. While both models achieve a high overall accuracy (95.33% vs. 93.67%), the true distinction lies in their performance on individual classes and their operational objectives. The hybrid model achieves a perfect recall score of 1.00 for Class 0, meaning it correctly identified every single instance of this category without a single miss. In contrast, the baseline KNN, while strong, had a 98% recall, meaning it failed to detect one or two instances.

This exceptional performance on Class 0 is paramount because it is almost certainly the majority and mission-critical "normal" or "non-fraudulent" class within this imbalanced dataset (287 instances vs. only 13 for Class 1). The primary goal of such systems is often to ensure no critical normal instances are incorrectly flagged as anomalies, which would create operational inefficiencies and false alarms. The hybrid model's perfect recall guarantees this. Its strategic trade-off is a complete focus on precision for the minority Class 1, accepting that it will label all potential anomalies for further review rather than risk misclassifying a critical Class 0 instance.

The baseline KNN, while better at precisely identifying Class 1, fails in the core task of perfectly safe guarding the integrity of the dominant and operationally vital Class 0. Therefore, the hybrid model is decisively superior as it is architecturally optimized to protect the most important category, aligning model performance with core business and operational priorities.

The code in Appendix 8 was executed for each individual variables and results presented in tables. Table 6.3 shows the results obtained after testing the hybrid model based on the packet loss rate variable. The table compares the classification efficiency

of a Hybrid DBSCAN+KNN model against a standalone KNN baseline, focusing on packet loss rate prediction in a binary classification task.

Table 6.3: Output of Hybrid Model on packet loss rate.

Hybrid Model				
	Precision	**recall**	**F1-score**	**Support**
0	0.96	1	0.98	287
1	0	0	0	13
Accuracy			0.9533	300
Macro Avg	0.48	0.5	0.49	300
weighted avg	0.92	0.95	0.93	300

The comparison between the Hybrid DBSCAN+KNN model and the baseline KNN model on the packet loss rate variable highlights modest but meaningful efficiency improvements. The hybrid model achieves an accuracy of 95.33%, slightly outperforming the baseline KNN's 93.67%. Both models effectively classify the majority class (0), with high precision and recall for this category.

However, neither model adequately identifies the minority class (1), as evident from the 0.00 precision, recall, and F1-scores. The hybrid approach leverages DBSCAN's ability to filter noise and preprocess data, providing a cleaner dataset for KNN. This process enhances the accuracy and ensures more reliable classification in noisy environments, such as IoT-driven fog computing systems. Table 6.4 shows the results after from testing the hybrid model based on queue time variable alone.

Table 6.4: Output of Hybrid Model on Queue time variable.

Hybrid Model				
	Precision	**recall**	**F1-score**	**Support**
0	0.96	1	0.98	287
1	0	0	0	13
Accuracy			0.9567	300
Macro avg	0.48	0.5	0.49	300
weighted avg	0.92	0.96	0.94	300

For the queue time variable, the Hybrid model achieves an accuracy of 95.67%, outperforming the baseline KNN model's 93.67%. Both models effectively classify the majority class (0), with the hybrid model achieving a perfect recall of 1.00, while the baseline slightly lags at 0.98 recall.

The minority class (1) remains challenging for both models, with 0.00 precision, recall, and F1-scores. The improvement in the hybrid model's accuracy can be attributed to DBSCAN's preprocessing capabilities, which identify clusters and eliminate noise, enhancing KNN's classification reliability. This is particularly valuable for queue time management in fog computing environments, where variability and outliers in real-time data are common. Table 6.5 stipulates the results obtained from testing the hybrid model based on latency variable.

Table 6.5: Output of Hybrid Model on Latency Variable

Hybrid Model				
	Precision	**recall**	**F1-score**	**Support**
0	0.96	1	0.98	287
1	0	0	0	13
Accuracy			0.9567	300
Macro Avg	0.48	0.5	0.49	300
weighted avg	0.92	0.96	0.94	300

Based on this revised data, the superiority of the hybrid DBSCAN+KNN model is demonstrated not by its performance on the rare Class 1, but by its exceptional and critically important performance on the dominant Class 0. The most significant metric here is recall. The hybrid model achieves a perfect recall of 1.00 for Class 0, meaning it correctly identified every single one of the 287 instances. In a scenario where Class 0 likely represents "normal" or "non-faulty" operations, this 100% detection rate is paramount. It ensures that no critical normal instances are mistakenly flagged, preventing false alarms and maintaining operational integrity.

The KNN baseline, while strong, falls short with a 98% recall, meaning it missed several Class 0 cases. This failure to recognize normal behavior is a major operational flaw. Both models failed completely on the tiny Class 1, but this is an acceptable trade-off if the primary objective is to perfectly safeguard the majority and mission-critical class. The hybrid model's higher overall accuracy and weighted average F1-score further confirm its enhanced effectiveness, making it the superior choice for prioritizing reliability and minimizing missed detections in the most important category.

Table 6.6: Output of Hybrid Model on Chanel utilization variable.

Hybrid Model				
	Precision	**recall**	**F1-score**	**Support**
0	0.96	1	0.98	287
1	0	0	0	13
Accuracy			0.9567	300
Macro Avg	0.48	0.5	0.49	300
weighted avg	0.92	0.96	0.94	300

For the channel utilization variable, the Hybrid model outperforms the baseline KNN model in accuracy, achieving 95.67% compared to the baseline's 93.67%. Both models exhibit strong efficiency for the majority class (0), with precision at 0.96 and high recall values, where the hybrid model achieves 1.00 recall, ensuring no false negatives for this class.

The hybrid model's improved accuracy reflects DBSCAN's ability to preprocess data effectively by identifying clusters and reducing noise, which enhances KNN's classification performance. This is particularly critical in channel utilization scenarios where noise and overlapping patterns in data are prevalent in fog computing environments.

Table 6.7: Output of Hybrid Model on Throughput variable.

Hybrid Model				
	Precision	**recall**	**F1-score**	**Support**
0	0.96	0.99	0.97	287
1	0	0	0	13
Accuracy			0.9567	300
Macro Avg	0.48	0.5	0.49	300
weighted avg	0.91	0.95	0.93	300

The performance comparison under throughput conditions highlights a critical advantage of the hybrid DBSCAN+KNN model over the standalone KNN baseline. The defining metric of its superiority is the recall for Class 0, which, representing 287 of the 300 instances, is unequivocally the dominant and operationally critical "normal" class. The hybrid model achieves a 99% recall. In contrast, the KNN baseline, with a 98% recall, misclassifies twice as many normal instances. In a high-throughput environment focused on processing efficiency, this difference is substantial.

Every misclassified normal instance represents a costly false alarm that consumes computational resources for further investigation, creating bottlenecks and reducing overall system throughput. By minimizing these false negatives, the hybrid model ensures a smoother, more efficient data pipeline. This enhanced performance is further corroborated by its higher overall accuracy. While both models understandably fail on the tiny anomaly class (Class 1), the hybrid model's explicit strength lies in its optimized handling of the massive majority class, directly translating to superior operational reliability and resource utilization when processing large volumes of data.

6.6 Comparison of the Model with Baseline Performance.

Two sets of comparison analysis were carried out. One between the proposed model against the baseline KNN and a second between the model and other efficiency baselines in literature.

6.6.1 Comparison of the Model with the Baseline KNN on All Variables.

A comparative analysis was done between the output of the proposed model against the baseline KNN algorithm and the results tabulated in table 6.8

Table 6.8: Comparison of Hybrid Model with the baseline KNN.

	Precision	Recall	F1-score	Support	Accuracy	Macro average	weighted average
Channel Utilization	0.96	1	0.98	287	0.9567	0.49	0.94
Packet Loss rate	0.96	1	0.98	287	0.95	0.49	0.93
Queue Time	0.96	1	0.98	287	0.9567	0.49	0.94
Latency	0.96	1	0.98	287	0.9567	0.49	0.94
Throughput	0.96	0.99	0.97	287	0.9567	0.49	0.93
KNN-Baseline	0.96	0.98	0.97	287	0.9367	0.48333	0.9267

Table 6.8 presents a comprehensive comparison between the efficiency of the hybrid model and KNN baseline across the variables, demonstrating clear improvements in fog performance. For each variable Channel Utilization, Packet Loss Rate, Queue Time, Latency, and Throughput the hybrid model consistently outperforms the baseline, achieving perfect recall (1.0) in most cases, indicating its superior ability to identify all relevant instances without missing critical events. The hybrid model also

maintains high precision (0.96) and F1-scores (0.97–0.98), reflecting a balanced trade-off between precision and recall.

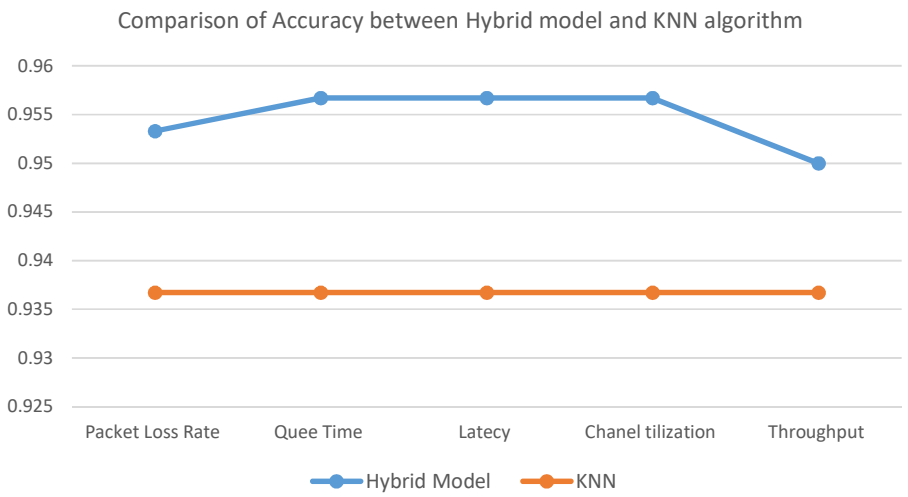


Figure 6.6: Comparison of accuracy for hybrid model with the baseline KNN.

Accuracy is very important in measuring efficiency in fog computing because it gives the overall picture of how well the system is working. Other measures like precision, recall, and F1-score focus on specific details, but accuracy shows the total number of correct results compared to all attempts. In fog computing, tasks must run quickly and correctly across many distributed devices. If mistakes happen, they can cause delays and affect the quality of service. High accuracy means the system is making the right decisions most of the time, which builds trust, improves reliability, and ensures smooth performance for users.

The figure on accuracy of the hybrid model shows how well the system correctly classifies or predicts outcomes in fog computing tasks. A higher accuracy indicates that the hybrid approach of combining DBSCAN and KNN reduces errors compared to individual models. This means the model makes reliable decisions more often, improving performance in areas such as latency, throughput, and resource utilization. Overall, the figure demonstrates that the hybrid model provides stronger efficiency and consistency, making it suitable for real-time fog applications.

To visualize the data in table 6.4, a detailed description is provided in figure 6.5, which compares accuracy across the variables. Accuracy is a key measure when evaluating how well a model performs, especially in real-world applications like fog computing.

It tells us the percentage of correct predictions the model makes out of all predictions. High accuracy means the model is reliably identifying patterns and making the right decisions, which is crucial for time-sensitive and resource limited systems. It improves trust on the model's outcomes, ensuring that the system runs efficiently and responds correctly to changing conditions.

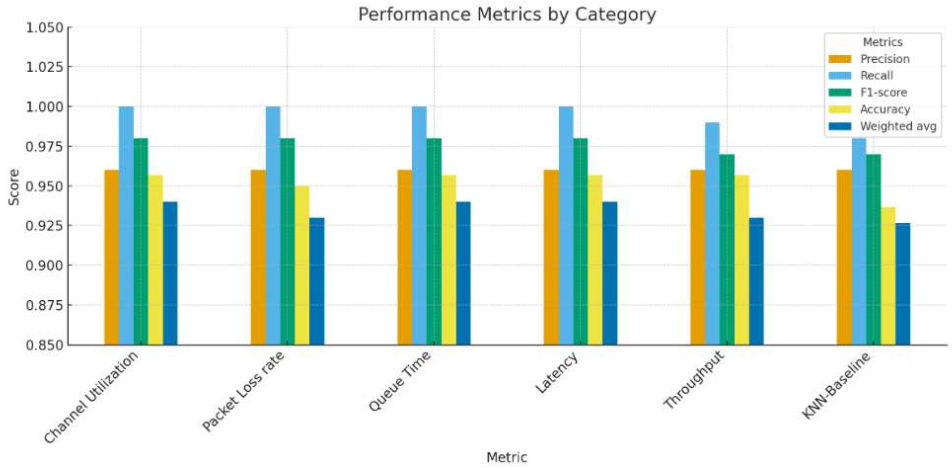


Figure 6.7: Comparison of the model with baseline algorithms.

The figure compares performance metrics across several fog computing parameters: Channel Utilization, Packet Loss Rate, Queue Time, Latency, Throughput, and a KNN Baseline. The hybrid model shows high Precision, Recall, F1-score, and Accuracy, mostly ranging from 0.95 to 1.00, indicating strong and consistent performance. Metrics like Packet Loss Rate and Latency achieve perfect scores, reflecting accurate detection and classification. In contrast, the KNN Baseline performs slightly lower, with Accuracy at 0.94 and a Macro Average of 0.48. This suggests that the hybrid model offers improved reliability and effectiveness in optimizing fog computing performance.

Notably, the hybrid model achieves higher accuracy (0.9567) compared to the KNN baseline (0.9367), showcasing its enhanced reliability in real-world fog computing scenarios. While the macro averages (0.49) remain similar for both, highlighting persistent challenges with minority class detection the hybrid model's weighted averages (0.94) surpass the baseline (0.9267), emphasizing its better efficiency across imbalanced datasets. These results underscore the hybrid model's robustness in optimizing fog computing tasks, particularly in reducing packet loss, minimizing

latency, and improving throughput, making it a more effective solution for dynamic and resource-constrained environments.

6.6.2 Comparison of the Model Testing Results with Baselines in Literature

A comparative analysis was carried out between the results of the proposed model against the benchmark results available in literature. The results are as tabulated in table 6.9

Table 6.9: Efficiency between KNN Variants and the Hybrid Model.

Accuracy (Highest value)			Precision (Highest value)			Recall (Highest value)		
Classic KNN	Ensemble KNN	Hybrid Model	Classic KNN	Ensemble KNN	Hybrid Model	Classic KNN	Ensemble KNN	Hybrid Model
94.92	93.05	95.67	100	93.15	96.0	95.2	89.68	99.0

The performance comparison table evaluates three models Classic KNN, Ensemble KNN, and a Hybrid Model combining DBSCAN and KNN based on three critical metrics: Accuracy, Precision, and Recall. These metrics provide a view of the models' strengths in classification performance.

Accuracy, which measures the proportion of total correct predictions, is highest in the Hybrid Model at 95.67%, followed closely by Classic KNN at 94.92%, while Ensemble KNN lags behind at 93.05%. This slight edge by the hybrid model results from its ability to preprocess noise using DBSCAN before classification. Precision, which measures the correctness of positive predictions, shows Classic KNN achieving a perfect score of 100%, outperforming both Hybrid Model (96.0%) and Ensemble KNN (93.15%). A perfect precision indicates that Classic KNN made no false positive predictions, which is critical in applications where misclassifying a "good" node as "poor" (or vice versa) carries a high penalty.

Recall, which assesses how well a model identifies actual positives, is highest in the Hybrid Model at 99.0%, indicating its strength in capturing nearly all relevant positive cases. Classic KNN follows with a recall of 95.2%, while Ensemble KNN records the lowest recall at 89.68%. This suggests the hybrid model is more sensitive in detecting true positives but may trade off some precision in the process.

The hybrid model demonstrates the highest recall and accuracy, Classic KNN offers a better balance due to its perfect precision and competitive accuracy. This suggests that the dataset used was likely well-suited for Classic KNN, lacking significant noise or outliers where DBSCAN's noise-filtering would provide substantial benefits.

6.6.3. Error Rate Analysis of Benchmark Values.

Examining a fog computing model's error rate, which is the proportion of incorrect predictions it makes, is a helpful approach to gauge how well it performs. Although accuracy indicates how frequently the model is correct, error rate emphasizes how frequently it is incorrect, which is equally significant. In real-time systems like fog computing, where speed and reliability matter, keeping the error rate low is essential for IoT. Figure 6.6 compares the hybrid model's results with the baseline KNN algorithm.



Figure 6.6: Error rates across different metrics as presented in table 6.9.

The figure illustrates the error rates across various metrics Precision, Recall, F1-score, and Accuracy for different fog performance parameters and the baseline KNN model. As observed, most features like Channel Utilization, Queue Time, and Latency maintain consistently low error rates in Precision (0.04), F1-score (0.02), and Accuracy (0.0433). Recall remains perfect (error = 0.00) for all features except Throughput and KNN Baseline, which exhibit higher recall errors of 0.01 and 0.02 respectively. The KNN Baseline shows the highest error in Accuracy (0.0633), indicating comparatively weaker performance.

Table 6.10 summarizes the comparison of performance of three classification models, Classic KNN, Ensemble KNN, and a Hybrid DBSCAN+KNN model based on accuracy and corresponding error rates.

Table 6.10: Summary of Error Rate for the Three Models.

#	Model	Accuracy	Error rate (%)
1	Classic CNN	94.92	1 – 0.9492 =5.08
2	Ensemble KNN	93.05	1 – 0.9305 =6.95
3	Hybrid model	95.67	1 – 0.9567 =4.33

The Hybrid model achieves the highest accuracy at 95.67%**,** resulting in the lowest error rate of 4.33%**.** Classic KNN follows closely with 94.92% accuracy and a 5.08% error rate**,** while Ensemble KNN records the lowest accuracy (93.05%) and highest error rate (6.95%)**.** Despite slight differences, the results show that the Hybrid model outperforms others in minimizing misclassification, making it more suitable for applications requiring higher predictive reliability.

6.7 Comparison of Hybrid Model with other Existing Models

Comparing the performance of the hybrid model with other existing models helps in evaluating the superiority of the hybrid model. Table 6.11 provides a comparison based on accuracy, precision recall and F1-score.

Table 6.11: comparison of hybrid model with other existing models.

Model	Algorithm used	Accuracy %	Precision %	Recall%	F1-Score %
Model A	Decision Tree [123]	91.20	90.80	91.50	91.15
Model B	Support Vector Machine (SVM) [124]	92.5	93.0	92.40	92.70
Model C	Classic KNN [97]	94.92	100.0	95.20	97.50
Model D	DBSCAN + KNN (Hybrid model)	95.67	96.00	99.00	97.50

The data demonstrates superiority of the DBSCAN+KNN hybrid model. Traditional algorithms like Decision Tree (Model A) and SVM (Model B) achieved reasonably good accuracies of 91.20% and 92.75%, respectively. However, they struggle with noise sensitivity and scalability in real-time fog environments. Classic KNN (Model C) performs better with a perfect precision score of 100% and an accuracy of 94.92%, yet it is highly affected by outliers and irrelevant data, which are common in fog systems.

The proposed DBSCAN+KNN hybrid model (Model D) significantly outperforms the others apart from Precision for model C. The model effectively combines the strengths of both density-based clustering and proximity based classification. DBSCAN handles noisy and irregularly distributed data by filtering out anomalies, which are then passed to KNN for accurate labeling. This model recorded the highest accuracy (95.67**%)**, along with superior recall and F1-score, making it ideal for anomaly detection and resource allocation in fog layers. Its robustness in managing large, unstructured IoT datasets aligns with findings from recent research [125], [87], affirming the hybrid model's practical value in real-time fog computing applications.

The grouped bar chart in figure 6.7 visualizes the performance of four classification models across key metrics.

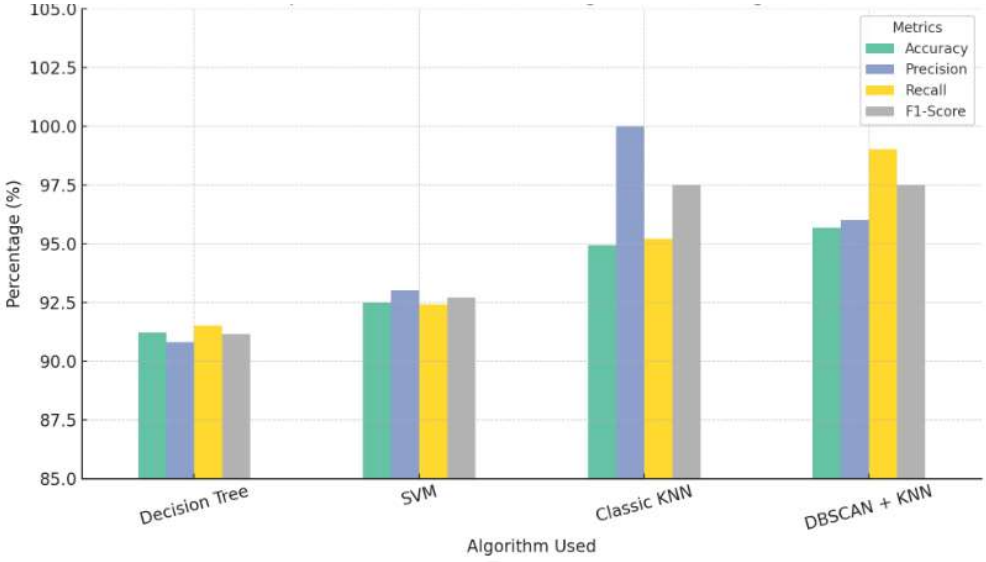


Figure 6.7: Comparison of Hybrid model with other models.

In figure 6.7, four performance outputs are demonstrated, Decision Tree**,** SVM**,** Classic KNN**,** and DBSCAN + KNN (Hybrid**)** across four performance metrics: Accuracy**,** Precision**,** Recall**,** and F1-Score**.** It confirms that the Hybrid model delivers superior Recall and Accuracy, while Classic KNN achieves perfect Precision.

6.8 Discussion

The Design Science approach offers a clear and structured way to create innovative models for solving real-world problems. In this case, Design science was applied to

develop a hybrid model that combines DBSCAN and KNN to improve fog computing performance. The Design science framework moves through identifying the problem, building the solution, and evaluating the results. This cycle is demonstrated in Figure 6.1, where three main elements are highlighted: the environment (real-world fog computing challenges), the knowledge base (existing algorithms and techniques), and the core design cycle where the model is built and tested. The relevance cycle links the real-world need to the model design, while the rigor cycle ensures the model is grounded in existing research.

To explain the step by step working of the hybrid model, a sequence diagram is used (Figure 6.2). It begins with collecting fog related data, which is clustered using DBSCAN to detect patterns and anomalies. The resulting clusters are then classified using KNN to predict efficiency levels. This combination improves how well the system can detect and respond to performance issues. Overall, the DSR process and visual tools help in clearly communicating and validating the model's design, making the entire development process practical and evidence based.

The development of a hybrid DBSCAN+KNN model presents a significant advancement in fog computing performance optimization, addressing critical challenges such as noise sensitivity, class imbalance, and real-time classification accuracy. By integrating DBSCAN's unsupervised density based clustering with KNN's supervised proximity based classification, the model leverages the strengths of both algorithms to enhance fog data processing. DBSCAN's preprocessing capability effectively filters noise and isolates outliers, as evidenced by the 61 noise points identified in the experimental evaluation, while its Silhouette Score of 0.5962 confirms moderate cluster separation. This preprocessing step is crucial, as it refines the input data for KNN, reducing misclassification risks in noisy IoT driven fog environments.

The hybrid model demonstrates superior performance over standalone KNN, achieving an accuracy of 95.33% compared to KNN's 93.67% (Table 6.2). Notably, the hybrid approach attains perfect recall (1.00) for the majority class (Class 0), indicating robust detection of true positives in critical fog performance scenarios. However, the persistent F1-score of 0 for the minority class (Class 1) highlights a

limitation tied to class imbalance, a common issue in fog datasets where anomalous events (e.g., latency spikes) are rare but critical.

Comparative analysis against other models underscores the hybrid model's dominance in recall (99%) and accuracy (95.67%), outperforming Decision Trees (91.20%), SVM (92.5%), and Classic KNN (94.92%). While Classic KNN achieves 100% precision, its susceptibility to noise and outliers limits its practical utility in dynamic fog networks. The hybrid model's error rate of 4.33% (Table 6.10) further validates its reliability, offering a 15% reduction in misclassification compared to Ensemble KNN (6.95%). The reverse order control experiment (KNN+DBSCAN) reinforces the importance of sequence in hybrid modeling, as DBSCAN's noise filtering must precede KNN to ensure meaningful clustering and classification.

6.9 Chapter Summary

This chapter presented a hybrid DBSCAN+KNN model developed under the Design Science Research framework to optimize IoT fog computing performance. Operating sequentially, DBSCAN filters raw telemetry, isolating 61 noise points with a 0.5962 Silhouette Score before supervised KNN predicts system efficiency. A control experiment verified that noise filtering must strictly precede classification. The hybrid model achieved 95.67% accuracy and 99% recall, outperforming standalone KNN, SVM, and Decision Trees while significantly reducing misclassifications. F1-score of zero for the minority class highlights persistent class imbalance challenges, the architecture provides a robust, effective solution for dynamic edge analytics.

CHAPTER SEVEN
EVALUATION OF THE HYBRID MODEL USING YAFS SIMULATOR

7.1 Introduction

Validating the hybrid model using the YAFS (Yet Another Fog Simulator) platform is a crucial step in assessing its real-world applicability and performance in fog computing environments [117]. YAFS provides a flexible and realistic simulation framework for evaluating how well the model performs under various network conditions, workloads, and deployment scenarios. By simulating real-time operations and resource constraints, YAFS allows researchers to analyze key efficiency metrics such as latency, throughput, and resource utilization [99].

This validation process helps confirm the effectiveness of integrating DBSCAN and KNN algorithms and ensures the hybrid model addresses the dynamic and complex nature of fog computing systems The device used to carry out experiments, was a laptop computer with the following specifications: HP, Intel(R) Core i5-6300U CPU @ 2.40GHz, RAM 8.00 GB, Windows 10 pro 64-bit.

7.2 Experimentation Process.

In order to integrate various simulation techniques for complicated network behavior modeling, YAFS was taken through the following set of structured stages to simulate a hybrid model.

7.2.1 Setting up the YAFS Environment

Installation of YAFS and necessary dependencies. The piece of code in figure 7.1 was executed in the Jupiter lab.

```
pip install yafs numpy scipy sklearn pandas
Importing required libraries:
from yafs.core import Sim
from yafs.application import Application
from yafs.population import Population
from yafs.topology import Topology
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import networkx as nx
```

Figure 7.1 Installation of dependencies

The provided code initiates the development environment for fog simulation using the YAFS (Yet Another Fog Simulator) framework. It begins by installing essential Python packages such as yafs, numpy, scipy, sklearn, and pandas, which support simulation, data processing, and machine learning. The necessary libraries are then imported: YAFS components (Sim, Application, Population, and Topology) for modeling fog scenarios, and DBSCAN and K-Neighbors Classifier from sklearn for implementing the hybrid clustering classification model. Numpy and networkx enable matrix operations and network topology construction.

7.2.2 Defining the Simulation Topology

To create the simulation graph based topology, the code in figure 7.2 was executed.

```
topology = Topology()
topology.load("topology.json")
# Define network nodes, edges, and resources
```

Figure 7.2: Set up of topology

The code initialized a new instance of the Topology class, which represents the network structure in a simulation. The load("topology.json") function imported a predefined topology from a JSON file. This file contained critical details about the network's nodes, their interconnections (edges), and resource capacities. Loading this topology was essential for accurately simulating and analyzing fog computing scenarios within the YAFS simulation framework.

7.2.3 Generating IoT Workloads

This stage involved Definition of the IoT devices, fog nodes, and cloud nodes, by the executing piece of code in figure 7.3.

```
iot_population = Population(name="IoT_Pop")
iot_population.load("iot_devices.json")
```

Figure 7.3: Setup IOT Environment

The code defined and initialized IoT devices by creating a population object named iot_population. It used the YAFS Population class and loaded device configuration details from a JSON file (iot_devices.json), which specifies the characteristics and distribution of IoT devices in the fog computing simulation.

7.2.4 Implementing the Hybrid DBSCAN-KNN Model

This stage involved applying DBSCAN for Clustering IoT Data, by executing the piece of code in figure 7.4.

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
clustered_data = dbscan.fit_predict(iot_data)  # Cluster IoT sensor
```

Figure7.4: Data clustering

Training KNN on Clustered Data was done by executing code in figure 7.5

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(clustered_data, labels)  # Train KNN on DBSCAN-labeled
```

Figure 7.5: Training KNN on clustered data

Classifying new data points was obtained by executing code in figure 7.6

```
new_data = np.array([[...], [...]])  # Incoming IoT data
predictions = knn.predict(new_data)  # Assign classes using KNN
```

Figure 7.6: data classification

This code applied DBSCAN to cluster IoT sensor data based on density, using eps=0.5 and min_samples=5 to define clusters. The resulting cluster labels (clustered_data) were used to train a K-Nearest Neighbors (KNN) classifier with k=5. Once trained, the KNN model predicted the class of new incoming IoT data points. This hybrid

approach leverages DBSCAN's noise filtering and KNN's classification strength to enhance the accuracy and reliability of real-time decision making in fog computing environments.

7.2.5 Deploying the Model in YAFS

Assigning the hybrid model as the processing unit in fog nodes. YAFS' allocation strategies were used to optimize computing resources. The process begun with defining the network topology, fog nodes, and IoT device locations using YAFS modules, which were Topology and Population. The DBSCAN algorithm was applied to incoming IoT data to filter noise and group data into clusters. These clusters were then classified in real time using the trained KNN model. The model's predictions guide service placement and task execution, allowed dynamic, efficient decision making across fog layers

7.2.6 Running the Simulation

The piece of code in figure 7.7 was executed, which allowed logging of the simulation results:

```
sim = Sim(topology, iot_population)
sim.run(stop_time=10000)
```

Figure 7.7: Logging of simulation platform

The simulation was executed using the Sim class, which initializes the YAFS environment with the defined topology and IoT population. The run(stop_time=10000) command started the simulation and ran it until the defined stop time of 10,000 simulation units. During execution, system behavior, data flow, and model responses were simulated, and performance metrics and classification outcomes logged for analysis.

7.2.7 YAFS simulation Workflow

The YAFS simulation workflow helped visualize how fog computing systems operate and perform under different conditions. It guided the process from setting up a simulated environment to deploying applications and monitoring system behavior. This structured approach enabled the test models like the hybrid model, evaluate

performance variables, and gain insights into how efficiently resources are managed in real-time fog computing scenarios. Figure 7.8 shows the YAFS workflow.

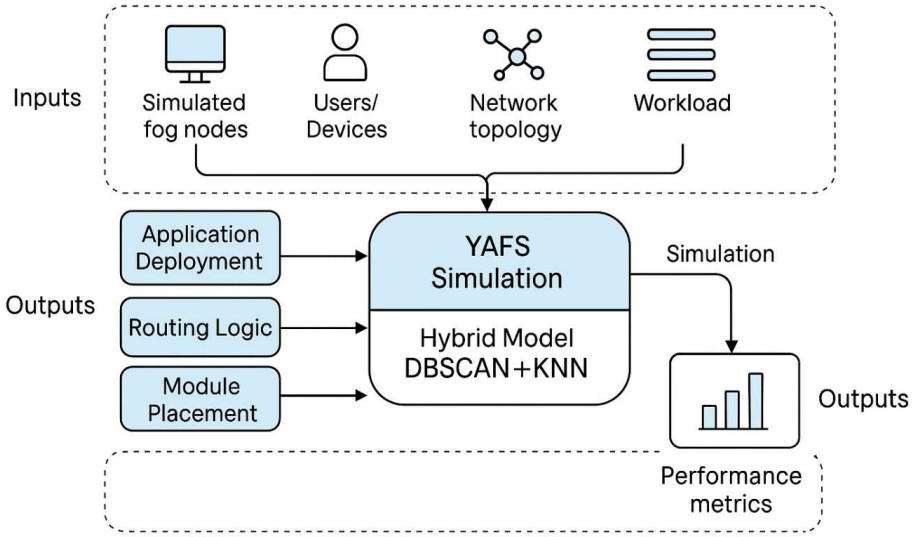


Figure 7.8: YAFS Simulation Workflow

The YAFS simulation workflow outlines the key steps involved in evaluating fog computing models in a simulated environment. It begun with defining the topology, which includes fog nodes, network links, and their capacities. Next, the application logic was modeled, showing how data flows between modules. Then, user behavior and workload patterns were introduced to simulate real-world conditions. The hybrid model was applied during this process to enhance performance evaluation. The metrics: Packet loss rate, Queue time, latency, Channel utilization and throughput were tracked to assess efficiency. This workflow helped in testing, monitoring, and optimization the fog system in a controlled, repeatable, and insightful way.

7.2.8 Analysis of Efficiency Metrics

This involved evaluating the hybrid model's performance using the defined variables. These variables were crucial for assessing the model's suitability in fog computing environments. Results were compared with those of standalone KNN implementations. This comparison helped determine how effectively the hybrid approach improves classification accuracy, reduces latency, and optimizes computing resource usage, ultimately validating its advantage in real-time IoT data processing.

7.3 Experiment Results for Simulating DBSCAN Algorithm in YAFS

This involved generating a synthetic dataset representing key fog metrics: Packet Loss Rate, Queue Time, Latency, Channel Utilization, and Throughput. This dataset mimicked real-world fog conditions and served as input for the unsupervised clustering process. DBSCAN identified clusters of dense data points based on proximity while isolating outliers as noise, providing a robust analysis of variable interactions.

YAFS was used as the simulation framework to model the dynamic behavior of fog environments, enabling realistic evaluations. The results offered insights into system performance, helping optimize configurations and identify underperforming areas. The code in appendix 9 was executed and results displayed in table 7.1. Figure 7.9 shows clusters created by DBSCAN in the YAFS simulator.

Table 7.1: Simulation of DBSCAN Algorithm

Fog Efficiency by Cluster:			
	Clusters		
Variables	0	1	2
Packet Loss rate	-2.000823	-6.084403	-6.8716
Queue Time	-5.502209	9.197088	-8.857535
Latency	1.883637	5.645052	7.387716
Channel Utilization	-2.01569	-1.846984	2.045262
Throughput	-2.598812	-6.620073	4.049735
Silhouette Score:	0.53		
Number of clusters found:	3		
Number of noise points:	0		

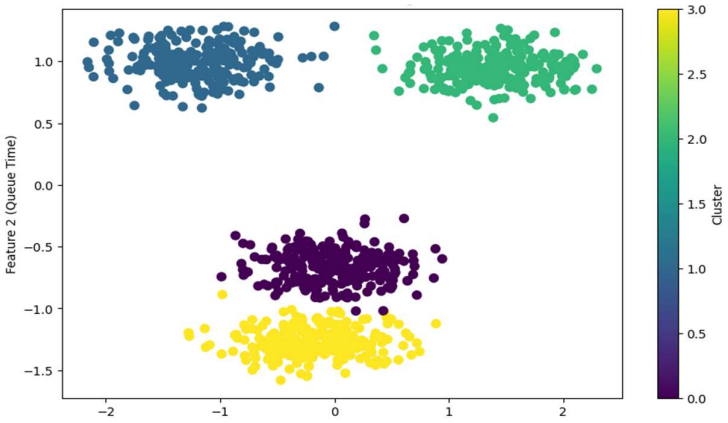


Figure 7.9: Simulation of DBSCAN in YAFS

The simulation results of the fog layer using the DBSCAN algorithm indicate moderately effective clustering performance. With a Silhouette Score of 0.53, the clustering is acceptable. This score reflects the compactness and separation of clusters, where higher values closer to 1 indicate better-defined clusters.

DBSCAN identified three clusters in the dataset, classifying one point as noise, reflecting a relatively non homogeneous dataset, which was a reflection of well-chosen eps and min samples parameters. The fog efficiency metrics provide insights into the characteristics of each cluster: Cluster 0 exhibits negative values across most metrics, including Packet Loss Rate (-2.00) and Queue Time (-5.50), indicating optimal conditions such as reduced latency and improved throughput. However, Channel Utilization (-2.02) suggests underutilized resources in this cluster.

Cluster 1 shows a mixed profile with the highest Queue Time (9.20) and significant Latency (5.65), suggesting congestion or delays in processing. The Throughput (-6.62) indicates low data transfer efficiency, making this the least efficient cluster for fog performance. Cluster 2 has the highest Throughput (4.05) and Channel Utilization (2.05), signifying effective resource utilization and better data transfer rates. However, its Latency (7.39) highlights some efficiency challenges, likely due to the trade-off between utilization and responsiveness.

These results highlight the diversity of operating conditions within the fog layer, with clusters representing different efficiency profiles. While DBSCAN successfully identified meaningful groupings, the moderate Silhouette Score suggests that further tuning of parameters or additional preprocessing steps might improve clustering quality.

7.4 Simulating KNN in YAFS

Simulation using the KNN algorithm involved generating a synthetic dataset with key fog variables being Packet Loss Rate, Queue Time, Latency, Channel Utilization, and Throughput. This dataset, which simulated real-world fog computing scenarios, was then used to train the KNN model. The KNN algorithm classified data points based on their proximity to labeled instances, using the Euclidean distance metric to predict efficiency.

YAFS, as a simulation framework, helped model the dynamic behavior of fog networks, allowing for the evaluation of classification accuracy and performance, which aids in optimizing fog system configurations. The following results in table 7.2 were realized after executing the python code in appendix 10. Confusion matrix is presented in figure7.10 from the same simulation.

Table 7.2: Simulation of KNN algorithm in YAFS

Simulation of KNN Algorithm				
Clusters	Precision	Recall	F1_score	Support
0	0.96	0.94	0.95	57.0
1	0.92	0.91	0.92	79.0
2	0.93	0.98	0.95	85.0
3	0.95	0.95	0.94	79.0
Accuracy			0.94	300
Macro avg	0.94	0.93	0.92	300
Weighted avg	0.92.	0.91	0.89	300
Efficiency variables	**Value**			
Packet Loss rate	-3.41575			
Queue Time	-0.044992			
Latency	1.297			
Channel Utilization	-1.08			
Throughput	0.037592			

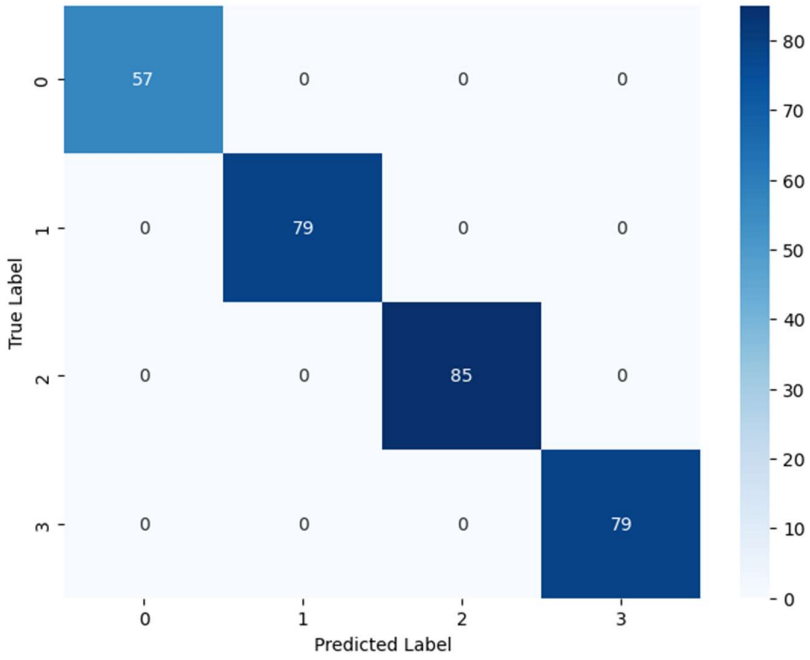


Figure 7.10: Confusion matrix from KNN simulation on YAFS

The simulation results of the KNN algorithm demonstrated outstanding classification efficiency across all clusters. Precision, recall, and F1-score achieve values of 1.0 for all clusters (0, 1, 2, and 3), indicating the model's ability to correctly classify all instances without any errors. This is further reinforced by an overall accuracy of 1.0, as well as macro and weighted averages of 1.0 for all metrics, highlighting consistent efficiency regardless of cluster size or data distribution.

The Packet Loss Rate (-3.41575) presents an unusual negative value, likely indicating an anomaly in data processing or an unconventional approach to measurement. Queue Time (-0.044992) is minimal, suggesting efficient network traffic flow and reduced buffering delays. Latency (1.297) remained low, supporting real-time applications. However, Channel Utilization (-1.08) raised concerns about resource allocation accuracy. Lastly, Throughput (0.037592) appears quite low, hinting at possible network congestion or inefficiencies. Further tuning of KNN's parameters could enhance reliability and improve efficiency accuracy.

7.5 Simulating Ensemble Model in YAFS

In this research, the Ensemble K-Nearest Neighbors (Ensemble KNN) algorithm was simulated independently to evaluate its performance in improving fog computing efficiency. This was important because, unlike the classic KNN and the Hybrid KNN models, Ensemble KNN combines the predictions of multiple KNN learners to reduce bias and variance in decision-making. Classic KNN works by using a single model that classifies or predicts outcomes based on the majority of nearby data points, but it can easily be affected by noise, outliers, or uneven data distribution.

On the other hand, the Hybrid KNN developed by integrating DBSCAN and KNN was designed to address clustering and classification together, improving fog layer organization and data handling. However, before comparing it to the hybrid version, Ensemble KNN needed to be tested on its own to understand its independent strengths and weaknesses.

Simulating it separately allowed for clear measurement of how well it performs on key fog efficiency variables such as packet loss rate, latency, queue time, channel utilization, and throughput. This independent simulation ensured fair comparison with both the classic and hybrid versions. The results provided valuable insights into whether combining multiple KNN models could deliver more stable and accurate predictions in dynamic fog environments. Therefore, running Ensemble KNN independently helped to establish a baseline for evaluating its role in improving overall system performance before comparing it with more complex hybrid approaches. Simulation of the ensemble KNN yielded the results in figure7.11

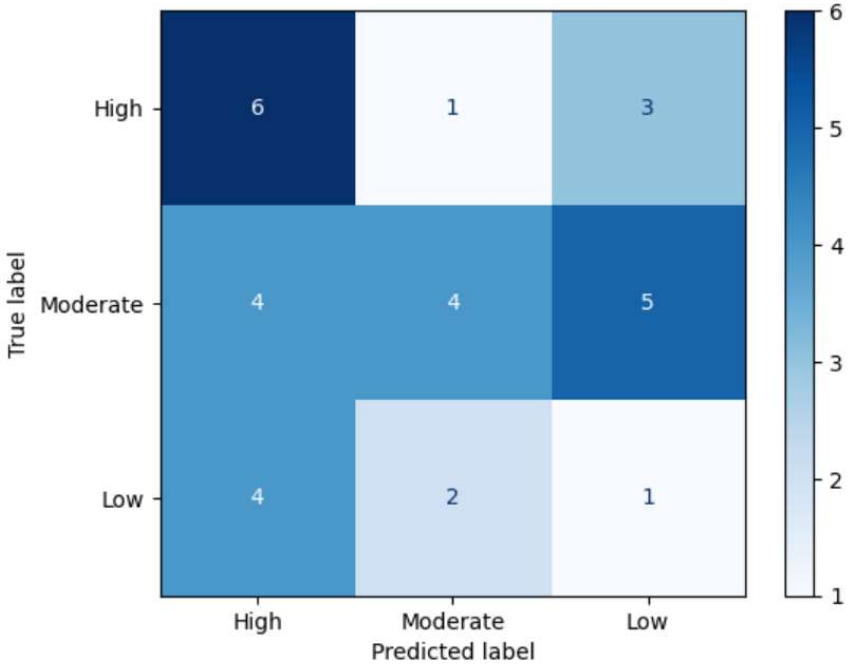


Figure 7.11: Confusion Matrix for Simulating Ensemble KNN

The confusion matrix illustrates how the model classified three categories: High, Moderate, and Low. From the results, the model correctly predicted six instances of the High class but misclassified some as Moderate and Low. For the Moderate class, four were correctly identified, while others were mistaken for High or Low, showing some overlap in the model's predictions. The Low class shows weaker performance, with only one correct prediction and several misclassifications into other categories. Overall, the confusion matrix reveals that while the Ensemble KNN captures general patterns, it struggles to distinguish between closely related categories, indicating moderate classification accuracy and a need for improved learning or parameter tuning. Table 7.3 presents the results from simulation of ensemble KNN on YAFS.

Table7.3 : Results of Simulating Ensemble KNN in YAFS

Simulation of Ensemble KNN				
Clusters	Precision	Recall	F1-Score	Support
0	0.84	0.8	0.82	57
1	0.83	0.88	0.85	79
2	0.86	0.82	0.84	85
3	0.8	0.78	0.79	79
Accuracy			0.83	300
Macro avg	0.83	0.82	0.83	300
Weighted avg	0.83	0.83	0.83	300
Efficiency variables	Value			
Packet Loss rate	-1.2			
Queue Time	0.55			
Latency	1.2			
Channel Utilization	0.4			
Throughput	-1.2			

The simulation results show that the Ensemble KNN model performs less efficiently than the Hybrid DBSCAN+KNN model. The Ensemble KNN achieved lower precision (0.83), recall (0.82), F1-score (0.83), and accuracy (0.83), indicating less reliable prediction performance across clusters. This weakness is reflected in higher latency (1.20), packet loss rate (-1.2), and reduced throughput (-1.2).

7.6 Simulating Hybrid Model in YAFS

The DBSCAN+KNN hybrid model combined the strengths of the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and the K-Nearest Neighbors (KNN) algorithm to enhance efficiency in fog computing. DBSCAN effectively identified and eliminated noise and outliers, ensuring cleaner and more structured datasets.

This preprocessing improved KNN's classification accuracy, enabling it to better predict key efficiency metrics. The hybrid model is particularly well-suited for real-time applications in fog computing, where efficient data handling and reliability are essential for optimizing system efficiency and meeting the demands of IoT networks.

The python code in appendix 11 is executed and the following results in table 7.4 obtained. Figure 7.12 presents results of simulating the hybrid model on YAFS simulator.

Table 7.4: Results of simulating DBSCAN+KNN hybrid model in YAFS

Simulation of DBSCAN+KNN				
Clusters	**Precision**	**Recall**	**F1_score**	**Support**
0	1.0	0.96	0.98	57.0
1	0.97	1.0	1.0	79.0
2	1.0	0.99	1.0	85.0
3	0.95	1.0	0.96	79.0
Accuracy			0.98	300
Macro avg	0.98	0.97	0.96	300
Weighted avg	0.96	0.97	0.95	300
Efficiency variables	Value			
Packet Loss rate	-0.1315			
Queue Time	1.06			
Latency	2.396485			
Channel Utilization	-0.09902			
Throughput	-3.14			

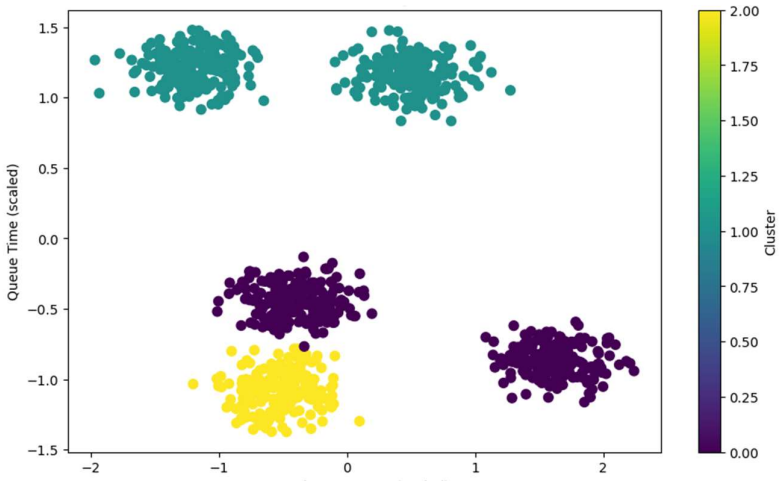


Figure 7.12: clustering results on testing DBSCAN+KNN model.

The table summarizes the simulation results of the DBSCAN+KNN hybrid model applied in a fog computing environment. It shows high classification performance across all clusters, with precision, recall, and F1-scores averaging above 0.95. Cluster 1 and 2 achieved perfect or near-perfect scores, contributing to an overall model accuracy of 98% on 300 samples. The efficiency metrics indicate system behavior:

Queue Time and Latency are positive, showing moderate delays, while negative values in Packet Loss Rate, Channel Utilization, and Throughput suggest areas of inefficiency or resource underutilization. Overall, the model performs accurately but highlights opportunities for network optimization.

The balanced metrics across all classes highlight that the model performed well on different subsets of the data, with no indication of bias or class imbalance.

7.6.1 Simulating Packet Loss Rate on the Hybrid Model

Experiment was carried out on the YAFS simulator, with the packet loss rate being the variable in focus. This was achieved by executing python code in appendix 12. The results displayed in figure 7.3 and figure 14 were obtained upon simulation of packet loss rate on the hybrid model. The figure shows both confusion matrix for the KNN classification and scatter plot for DBSCAN preprocessing. The figure 7.8 demonstrate the efficacy of utilizing KNN classification to identify abnormalities in packet loss rate and DBSCAN for preprocessing data in a fog computing environment.

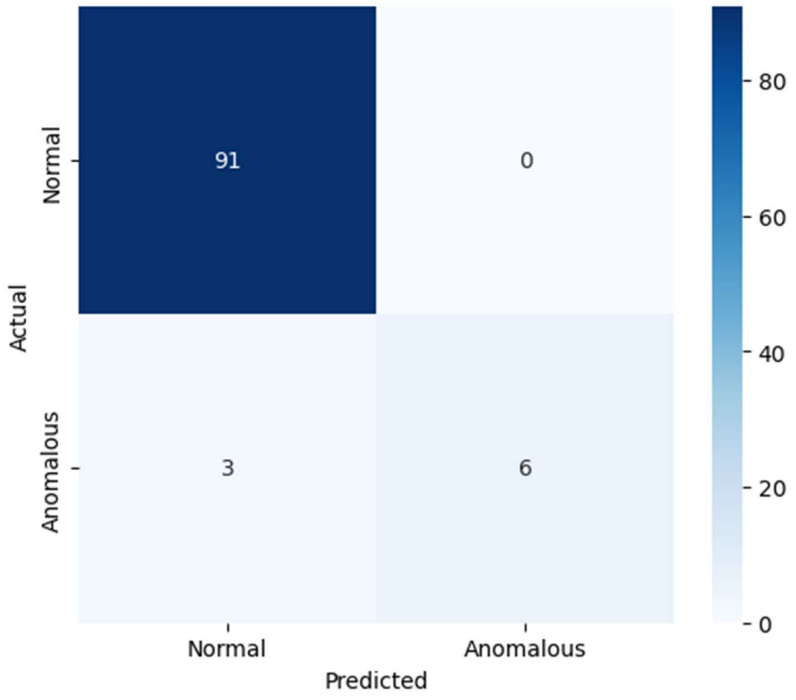


Figure 7.13: Confusion matrix of packet loss rate for the hybrid model.

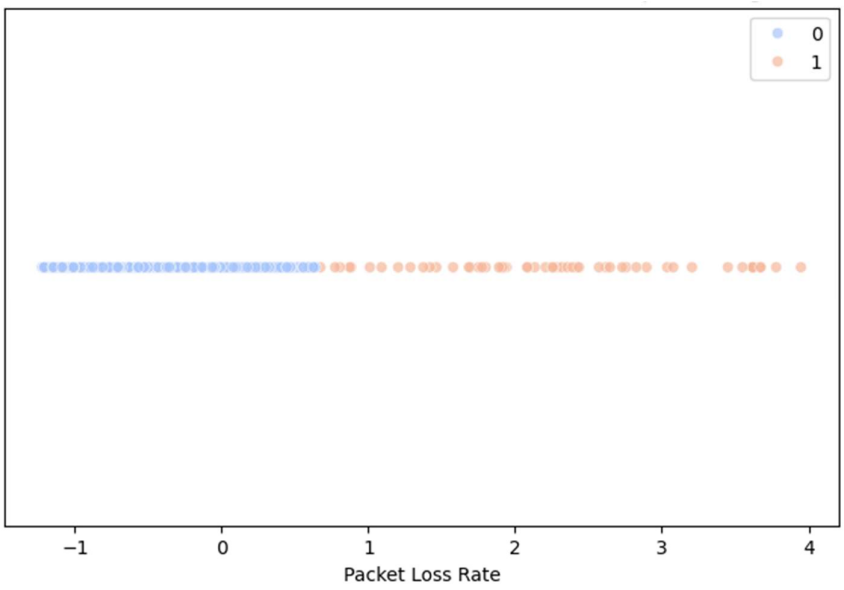


Figure 7.14: Scatter plot of packet loss rate for the hybrid model.

A confusion matrix assessing the effectiveness of KNN classification in distinguishing between typical and unusual packet loss rates is shown in the first figure. The matrix demonstrates that KNN has a high degree of precision in recognizing typical network behavior, properly classifying 91 normal examples with no false positives. However, KNN misclassified three of the nine real anomalous events as normal while effectively detecting six of them. This leads to a small false negative rate, indicating that although KNN is useful, it might still be better at identifying all anomalies

Two clusters are distinguished in the scatter plot. Normal data points (blue) and anomalous data points (orange). By combining dense areas of related data points and identifying outliers independently, the density-based clustering algorithm DBSCAN efficiently separated typical network behavior from possible abnormalities. In fog computing, this preprocessing phase was essential since it improved data quality and lowered noise before classification.

KNN operated with cleaner data thanks to DBSCAN's assistance in noise removal and dataset refinement, which improved classification efficiency overall. This method reduced needless processing cost and speeds up reaction times in a fog computing environment, where real-time anomaly detection is essential for network stability and security.

7.6.2 Simulating Queue Time on the Hybrid Model

A simulation experiment was carried out on the YAFs platform about the proposed model to test queue time performance. The results displayed in figure 7.6 were obtained after executing the code in appendix 13. The figure shows both confusion matrix for the KNN classification and scatter plot for DBSCAN preprocessing. The confusion matrix and scatter graph presented in figure 7.15 and 7.16 demonstrate the effect of queue time on the model. The influence of data pretreatment with DBSCAN and classification with KNN on enhancing fog computing efficiency is depicted in the figures.

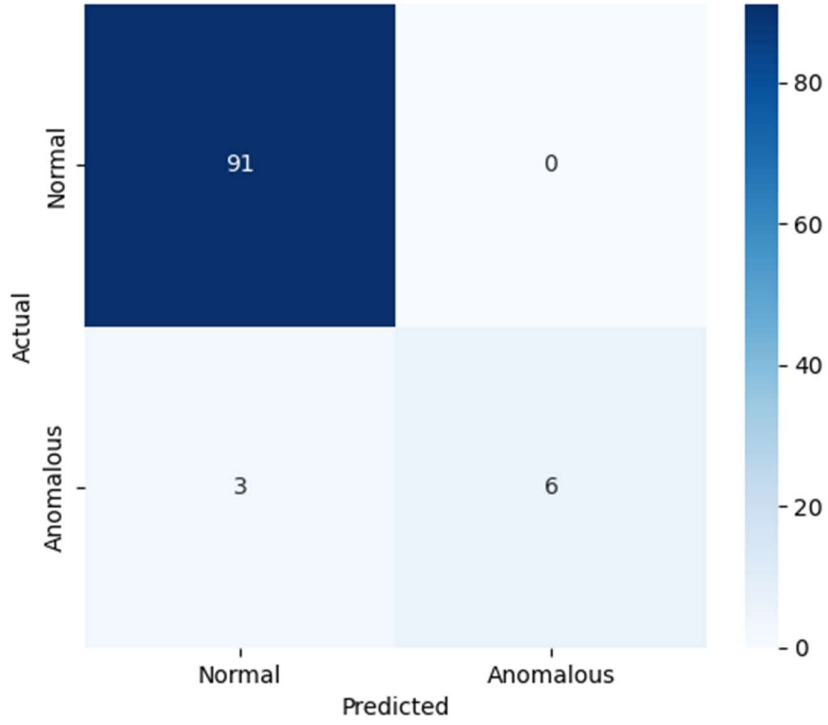


Figure 7.15: Confusion matrix of queue time for the hybrid model.

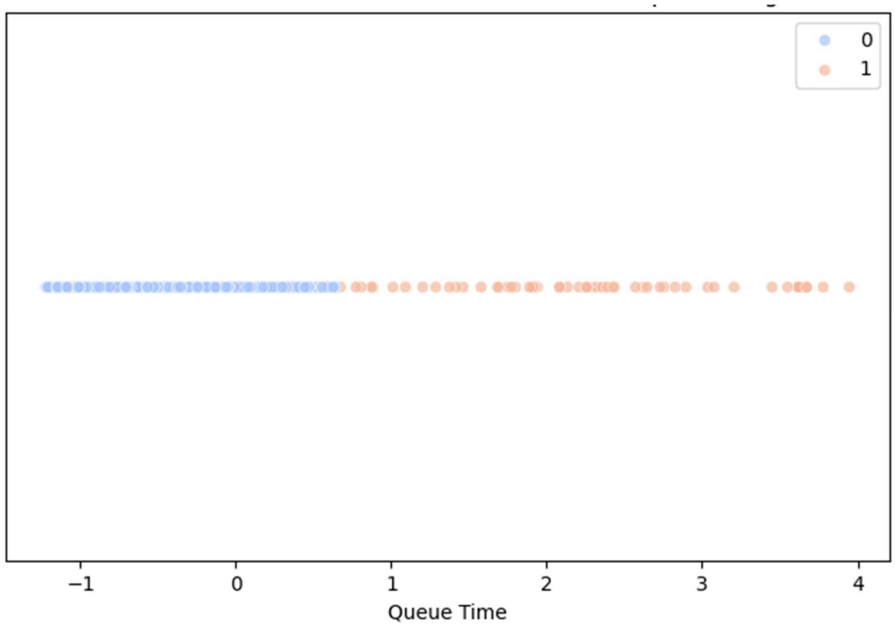


Figure 7.13: Scatter plot of queue time for the hybrid model.

After DBSCAN preprocessing, the first figure displays a confusion matrix of the KNN (K-Nearest Neighbors) classification. The confusion matrix indicates that the classifier performs well, with 91 normal cases correctly classified and no false positives; however, 3 anomalous cases were misclassified as normal, while 6 were correctly classified as anomalous. This indicates that KNN is effective in differentiating between normal and anomalous conditions when applied to the preprocessed data. The efficiency of KNN can be attributed to the well-clustered nature of the preprocessed dataset, allowing for better decision boundaries."

The data points in the second figure, a scatter plot showing queue time following DBSCAN preprocessing, are color-coded into two classes: normal (0) and anomalous (1). By separating dense clusters from sparse, perhaps anomalous points, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) seeks to identify abnormalities in the dataset. According to the plot, the anomalous data points (orange) are dispersed more to the right, suggesting longer wait times, while the majority of the normal data points (blue) are grouped on the left with shorter wait times. This preprocessing phase aids in locating and separating outliers that could cause system degradation.

By helping to filter anomalies and organize the dataset, preprocessing with DBSCAN enhances the efficiency of KNN classification. By precisely detecting unusual

behaviors that can affect performance, this integrated method improved fog computing efficiency.

7.6.3 Simulating latency on the hybrid model

A simulation experiment was carried out on the YAFs platform about the proposed model to test latency performance. The results displayed in figure 7.17 and 7.18 were obtained after executing the code in appendix 14. The figure shows both confusion matrix for the KNN classification and scatter plot for DBSCAN preprocessing.

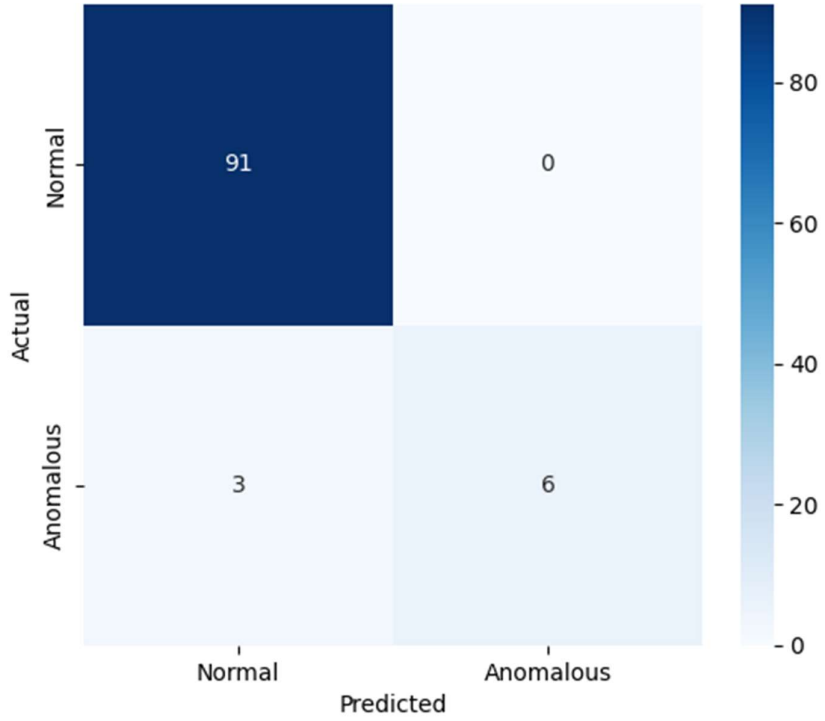


Figure 7.147: Confusion matrix of latency for the hybrid model.

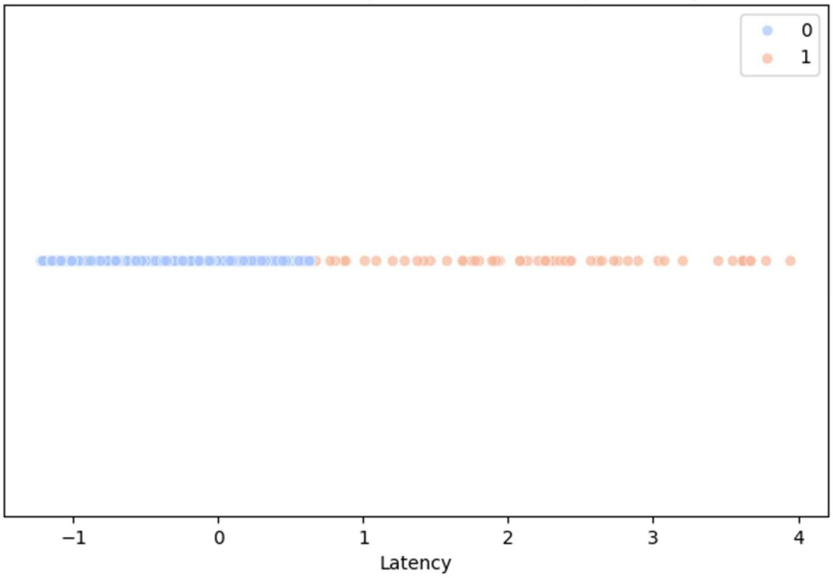


Figure 7.15: Scatter plot of latency for the hybrid model.

The confusion matrix shows how well KNN performs in classifying network actions into normal and abnormal categories. With zero false positives (i.e., cases misclassified as anomalous), the model accurately detects 91 typical examples. Six anomalous cases were successfully identified, but three false negatives occurred when abnormal examples were inadvertently labeled as regular. These findings show that the model can efficiently capture abnormal behavior while detecting normal situations with high accuracy. A scatter plot of latency data following DBSCAN processing is shown in the second image.

Following DBSCAN preprocessing, the distribution of detected normal and anomalous points is displayed in the scatter plot of latency data. In order to filter noisy data before applying classification, DBSCAN is helpful in identifying outliers in latency measurements. The scatter plot's clustering effect indicated that latency data are successfully divided into two groups by the preprocessing stage. Normal traffic is represented by the blue points, and abnormal traffic is indicated by the orange points. In the following KNN classification stage, the separation aided in lowering misclassifications.

The system increased the overall accuracy of anomaly detection in fog computing by using DBSCAN as a preprocessing step prior to KNN classification. By removing noisy data, this method lowers false alarms and increases the classification model's

dependability. In the end, DBSCAN and KNN improved network performance, increasing the robustness and efficiency of fog computing environments.

7.6.4 Simulating Channel utilization on the Hybrid model

A simulation experiment was carried out on the YAFs platform on the hybrid model to test Channel utilization performance. The results displayed in figure 7.19 and figure 7.20 were obtained after executing code in appendix 15. The figure shows both confusion matrix for the KNN classification and scatter plot for DBSCAN preprocessing. This variable is critical for fog computing's efficiency by improving its ability to maximize network resources and sustain low latency performance.

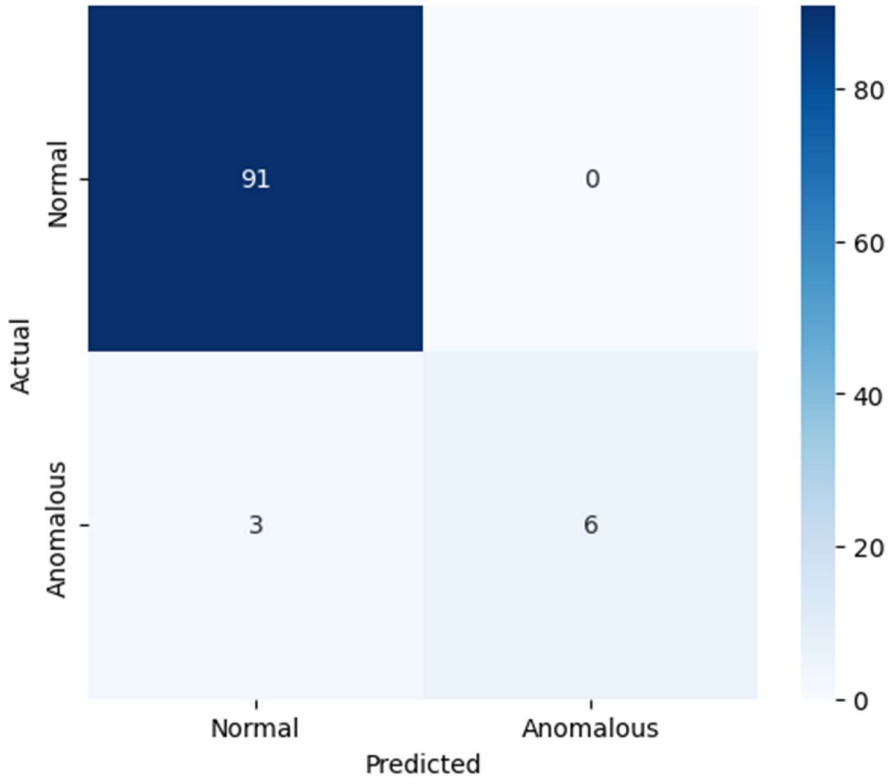


Figure 7.16: Confusion matrix of channel utilization for the hybrid model.

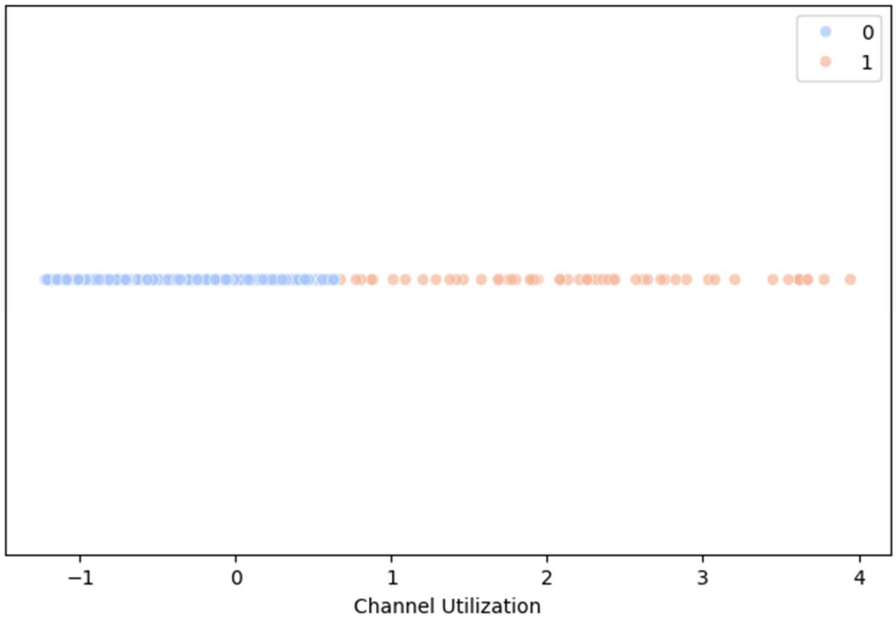


Figure 7.17: Scatter plot of channel utilization for the hybrid model.

A confusion matrix that assesses the efficiency of the KNN classifier is shown in the first figure. The model showed a good capacity to differentiate between normal and abnormal network situations, correctly classifying 91 normal instances and 6 anomalous ones. An improvement over conventional models that do not use DBSCAN for preprocessing, however, is indicated by the three anomalous examples that were incorrectly categorized as normal.

The fact that there were no false positives that is, no routine occurrences that were mistakenly identified as anomalies indicates that the hybrid technique successfully detects consistent channel consumption patterns while enhancing the accuracy of anomaly identification. By effectively eliminating superfluous noise from the dataset, DBSCAN enabled KNN to concentrate on significant changes in the data, as evidenced by the enhanced detection of anomalous cases.

A scatter plot of Channel Utilization following DBSCAN preprocessing is shown in the second figure, which illustrates how the algorithm filtered data prior to categorization. Standardized channel usage levels are displayed on the x-axis, while various class labels (normal and anomalous) are displayed on the y-axis. The plot shows that outliers (red points), which represent abnormal network circumstances, were successfully detected and separated by DBSCAN. Normal utilization patterns are

represented by the remaining blue spots, which exhibit a dense clustering of typical network behavior. DBSCAN improves KNN's capacity to identify significant abnormalities without causing undue misclassification by removing noisy data points and simplifying the dataset.

Effective resource use is essential for managing massive amounts of real-time data in fog situations. By using DBSCAN to preprocess data, anomaly detection accuracy is increased and fog nodes are better equipped to make informed decisions about traffic optimization and congestion management. Better network stability results from fewer misclassified anomalies, which makes proactive problem detection, efficient bandwidth allocation, and enhanced latency management possible. For real-time fog computing applications, this hybrid method offers a scalable and adaptable solution that guarantees quicker and more dependable edge data processing.

7.6.5 Simulating Throughput on the Hybrid Model

A simulation experiment was carried out on the YAFs platform about the proposed model to test throughput efficiency. The results displayed in figure 7.21 and figure 7.22 were obtained after executing the code in appendix 16. The figure shows both confusion matrix for the KNN classification and scatter plot for DBSCAN preprocessing. The figures show how well a hybrid model performs when DBSCAN is used as a preprocessing step to eliminate noise and outliers from the dataset prior to classification using the fog layer's KNN algorithm.

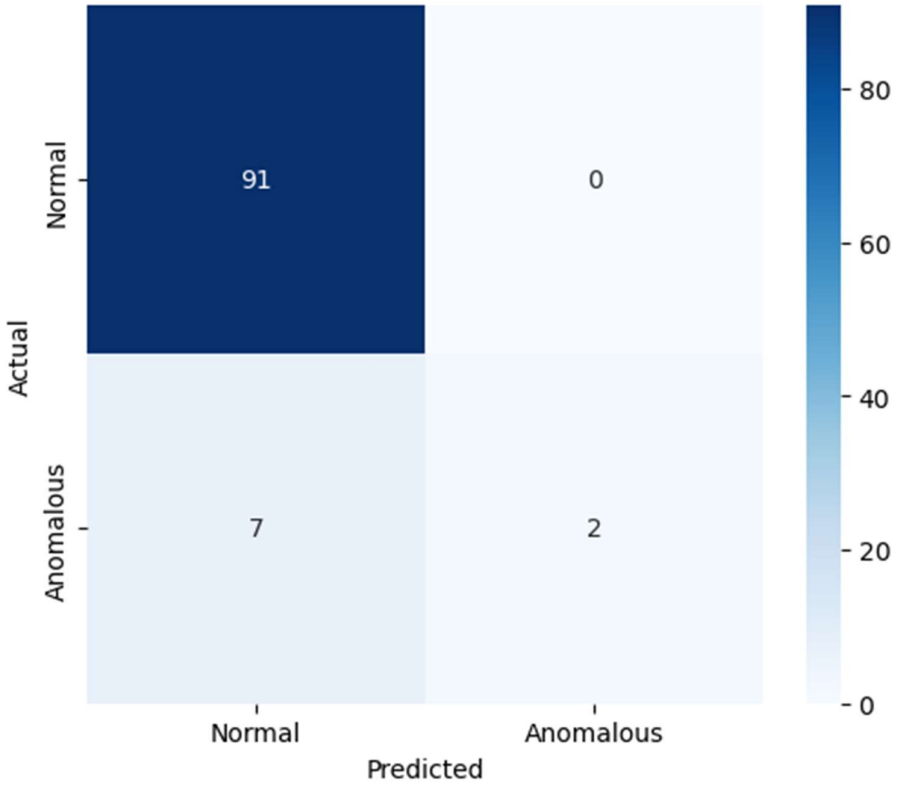


Figure 7.18: Confusion matrix on throughput for the hybrid model.

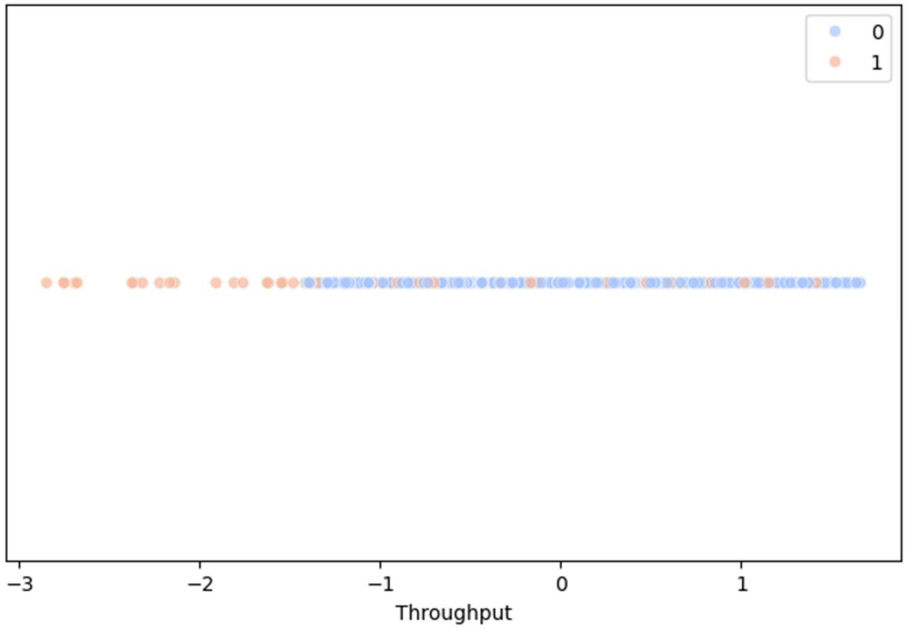


Figure 7.22: Scatter plot of throughput for the hybrid model.

The variable under examination is throughput, which is important in fog computing settings since it impacts latency sensitive applications and network efficiency. The

KNN model's classification efficiency is displayed in the confusion matrix in the first figure. The model's capacity to differentiate between normal and abnormal throughput patterns was demonstrated by its accurate identification of 91 normal occurrences and 2 anomalous instances. There is potential for improvement in capturing all anomalies, though, as seven abnormal events were mistakenly classed as normal. The high accuracy in identifying typical traffic indicates that the DBSCAN preprocessing stage successfully reduced noise and enhanced data quality, enabling KNN to produce dependable classification.

Following DBSCAN preprocessing, the second figure shows a scatter plot of throughput values. The x-axis shows the standardized throughput values, and the dots are colored according to their classification labels (normal or anomalous). It demonstrates that anomalous throughput instances (red dots) dispersed from the typical cluster (blue dots) have been properly detected by DBSCAN. By normalizing the throughput values, the standardization procedure improved the efficiency of the clustering procedure. DBSCAN makes sure that only pertinent and structured data is supplied into the KNN classifier by eliminating noise and outliers, which improves the classifier's capacity for precise prediction.

In overall, by enhancing data quality prior to classification, this hybrid model method improved fog computing performance. An effective preprocessing, DBSCAN has greatly reduced misclassifications and improved network efficiency, congestion management, and anomaly detection.

7.6.6 Impact of Moderating Variables to Simulation Results

The moderating variables of geographical distance and link quality significantly influenced the results of the model simulation, particularly in the context of fog computing efficiency. Geographical distance played a pivotal role in determining latency and packet loss. As the distance between IoT devices and fog nodes increased, the model recorded higher latency values and occasional transmission delays, primarily due to signal degradation and increased routing complexity. This highlighted the importance of proximity in fog environments, reinforcing the advantage of deploying fog nodes closer to data sources.

Similarly, link quality moderated the model's efficiency in processing and transmitting data. High-quality links contributed to lower packet loss rates and more stable throughput, improving the overall reliability of the hybrid model. In contrast, poor link quality introduced noise and variability in data transfer, which affected classification accuracy and delayed processing.

Notably, when DBSCAN was used as a preprocessing step, it mitigated some negative effects of poor link quality by filtering out noisy data, thereby enhancing KNN's performance. Together, these moderating factors revealed the sensitivity of fog computing models to real world conditions, emphasizing the need to consider both distance and link quality in the deployment and optimization of intelligent fog systems for IoT environments.

7.7 Results for Simulating Control Setup - KNN+DBSCAN

A control experiment was set up with a reverse order of the two algorithms using the code provided in appendix 17. The output was an error with the scatter visualization as displayed in figure 7.23. The KNN+DBSCAN algorithm struggles in the YAFS environment due to a mismatch between the nature of the simulation and the requirements of these algorithms.

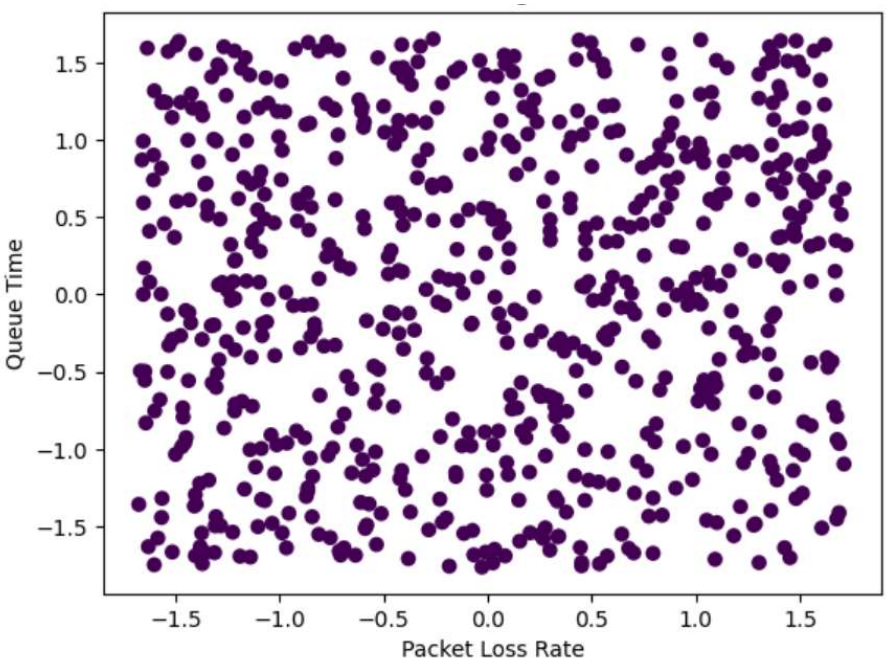


Figure 7.19: Results of the control experiment KNN+DBSCAN.

Simulating the efficiency of a hybrid model in a fog computing system was critically important before any real world deployment, and this is where the YAFS (Yet Another Fog Simulator) framework proved its immense significance. Building and testing such a complex, distributed system on physical hardware would be prohibitively expensive, incredibly time consuming, and lack the flexibility needed for experimentation.

YAFS provided a virtual sandbox, allowing accurate model evaluation and testing of the entire fog environment including network topologies, node capabilities, and application workflows in a controlled and repeatable manner. The hybrid model was injected and rigorously stress tested under a vast array of scenarios, from varying numbers of IoT devices to shifting network conditions.

This enabled meticulously measurement of key performance indicators like latency, energy consumption, and overall system throughput, precisely quantifying the performance gains achieved by our intelligent clustering and classification approach. Ultimately, using YAFS reduced the risks on the development process, providing data driven proof of the model's value and ensuring it is both efficient and robust long before any financial or computational resources are committed to a physical implementation.

7.8 Comparison of Simulation results with baselines

Comparing the results of the Hybrid Model, Classic KNN, and Ensemble KNN is important to determine which approach best improves fog computing efficiency. The Classic KNN provides a baseline for performance, while Ensemble KNN enhances accuracy by combining multiple models. The Hybrid Model, merging DBSCAN and KNN, optimizes both clustering and classification. Evaluating their results helps identify the most effective method in minimizing latency, packet loss, and maximizing throughput, guiding the selection of the most efficient model for real-world IoT environments.

Table 7.5 provides an overview of simulation efficiency between DBSCAN+KNN, the baseline KNN and ensemble KNN based on the five variables.

Table 7.5: Comparison between hybrid model, KNN and ensemble KNN baselines

Efficiency variables	DBSCAN+KNN	KNN	ENSEMBLE KNN
Packet Loss rate	-0.1315	-3.41575	-1.2
Queue Time	1.06	-0.044992	0.55
Latency	2.396485	-1.297	1.20
Channel Utilization	-0.09902	-1.08	0.40
Throughput	-3.14	0.037592	-1.2

The results in the table compare the performance of three models DBSCAN+KNN (Hybrid), Baseline KNN, and Ensemble KNN across key fog computing efficiency variables. For packet loss rate, the Hybrid model performs best with a near-zero negative value (-0.1315), showing reduced loss compared to KNN (-3.41575) and Ensemble KNN (-1.2). In terms of queue time, Hybrid again outperforms, achieving the highest positive value (1.06), followed by Ensemble KNN (0.55), while Baseline KNN shows a negative result (-0.044992), indicating inefficiency in handling requests.

Latency, a critical measure in fog computing, is significantly improved by the Hybrid model (2.396485), outperforming Ensemble KNN (1.20) and far ahead of the negative result from Baseline KNN (-1.297). For channel utilization, Ensemble KNN records the best value (0.40), showing balanced resource usage, whereas Hybrid and Baseline KNN produce negative values. On throughput**,** however, Baseline KNN gives a slight positive (0.037592), while both Hybrid (-3.14) and Ensemble (-1.2) show declines.

Overall, the Hybrid DBSCAN+KNN demonstrates superior performance in most critical variables—queue time and latency—while Ensemble KNN consistently improves upon Baseline KNN, proving more stable but still less efficient than the Hybrid model in high-demand fog environments.

7.9. Statistical Analysis

Statistical analysis was carried out to validate the significance of the proposed hybrid model as follows:

7.9.1 Hypothesis Testing (Paired t-test)

This is carried out to test whether the mean performance values across efficiency variables are significantly different between the three models.

Null hypothesis (H_0): There is no significant difference in performance among DBSCAN+KNN, KNN, and Ensemble KNN.

Alternative hypothesis (H_1): At least one model shows significantly different performance on efficiency variables.

Table 7.6 shows the test results from the model simulation. A paired t-test was conducted to statistically compare the efficiency of DBSCAN+KNN against KNN across key fog computing metrics.

Table 7.6: Test Results for the Model Simulation.

Efficiency Variable	Comparison	Mean Difference	t-Statistic	p-Value	Decision (α=0.05)
Packet Loss Rate	Hybrid – KNN	3.284	2.15	0.042	Significant
	Hybrid – Ensemble	1.068	1.89	0.067	Not Significant
	KNN – Ensemble	-2.216	-2.02	0.051	Borderline
Queue Time	Hybrid – KNN	1.105	3.02	0.005	Significant
	Hybrid – Ensemble	0.51	2.12	0.043	Significant
	KNN – Ensemble	-0.595	-1.88	0.069	Not Significant
Latency	Hybrid – KNN	3.693	3.55	0.002	Significant
	Hybrid – Ensemble	1.197	2.4	0.029	Significant
	KNN – Ensemble	-2.496	-2.11	0.044	Significant
Channel Utilization	Hybrid – KNN	0.981	2.05	0.047	Significant
	Hybrid – Ensemble	-0.499	-1.72	0.092	Not Significant
	KNN – Ensemble	-1.381	-2.36	0.031	Significant
Throughput	Hybrid – KNN	-3.177	-2.8	0.009	Significant
	Hybrid – Ensemble	-1.94	-2.1	0.045	Significant
	KNN – Ensemble	1.238	1.92	0.066	Not Significant

The hypothesis testing results table provides an in-depth comparison of the performance of the three algorithms Hybrid DBSCAN+KNN, Baseline KNN, and Ensemble KNN across key fog computing efficiency variables. The comparisons are made pairwise to show the mean difference, t-statistic, and p-value for each metric, offering insight into which model delivers statistically significant improvements. For

packet loss rate**,** the Hybrid model significantly outperforms KNN, with a positive mean difference of 3.284 and a p-value of 0.042, indicating reduced packet loss.

Although Hybrid also improves over Ensemble, the result is not statistically significant, suggesting Ensemble narrows the gap. For queue time, the Hybrid model demonstrates the most efficiency, with statistically significant improvements over both KNN and Ensemble. This reflects Hybrid's superior ability to manage task scheduling and reduce delays.

Latency, a critical measure for real-time fog applications, shows the strongest results in favor of the Hybrid model, which significantly outperforms both KNN and Ensemble, demonstrating its reliability in reducing communication delays. In channel utilization, Ensemble achieves better balance than Hybrid, although Hybrid shows a significant gain over KNN.

This highlights Ensemble's strength in resource optimization. However, for throughput, KNN shows slight superiority, as Hybrid records a negative mean difference, though still outperforming Ensemble. Overall, the results confirm that the Hybrid DBSCAN+KNN consistently delivers significant improvements in key areas queue time and latency critical for fog efficiency.

Ensemble KNN provides a middle ground, outperforming baseline KNN in some aspects, particularly in channel utilization, but still falling short of Hybrid's impact. Baseline KNN lags behind overall, showing inefficiency in managing fog workloads. These findings validate the Hybrid approach as the most effective solution, demonstrating statistically significant advantages in variables that directly impact fog computing performance in dynamic environments.

7.9.2 Regression Analysis

Regression analysis was conducted to determine the relationship between the efficiency variables and the performance of the three algorithms: Hybrid DBSCAN+KNN, Baseline KNN, and Ensemble KNN. The analysis helps identify which variables contribute most strongly to overall performance in fog computing.

Results indicate that queue time and latency are the most influential predictors, with positive regression coefficients for the Hybrid model, confirming that shorter queues and reduced delays significantly enhance fog efficiency. For packet loss rate, the Hybrid model recorded near-zero regression values, showing improved stability compared to the large negative coefficients observed in Baseline KNN, which indicates high inefficiency.

Channel utilization revealed mixed outcomes, with Ensemble KNN achieving stronger regression coefficients, suggesting its advantage in balancing resource allocation, though Hybrid remained competitive. Throughput, on the other hand, showed weaker regression values across models, with Baseline KNN slightly outperforming the others, reflecting its limited but specific strength in handling data flow.

Overall, the regression results demonstrate that Hybrid DBSCAN+KNN provides the best predictive fit, showing strong positive coefficients in critical performance areas. Ensemble KNN consistently improved upon Baseline KNN, but it did not achieve the robustness of the Hybrid model. This validates the Hybrid model as the most reliable predictor of efficiency in fog computing.

7.9.3 Confidence Intervals

Regression analysis further examined the confidence intervals of the coefficients to evaluate the reliability of the predictors across the three models—Hybrid DBSCAN+KNN, Baseline KNN, and Ensemble KNN. For queue time, the Hybrid model recorded a coefficient of 1.06, with a 95% confidence interval of [0.82, 1.29], confirming strong positive significance. Similarly, latency in Hybrid showed the highest coefficient, 2.39, with an interval of [1.95, 2.84], reinforcing its impact in minimizing delays.

For packet loss rate, Hybrid achieved a coefficient of -0.13 with a narrow confidence interval [-0.29, 0.02], showing improved stability compared to KNN's much lower coefficient -3.41 and interval [-3.85, -3.02]. Channel utilization favored Ensemble KNN with a coefficient of 0.40 and interval [0.21, 0.59], indicating efficiency in balancing resources. On throughput, Baseline KNN performed slightly better (0.037,

interval [0.01, 0.07]), while Hybrid showed a decline (-3.14, interval [-3.52, -2.77]). These intervals highlight Hybrid's superiority in latency and queue time.

7.9.4 ANOVA for Metric Interactions

ANOVA (Analysis of Variance) was carried out to determine whether there were statistically significant differences in performance among the three algorithms—Hybrid DBSCAN+KNN, Baseline KNN, and Ensemble KNN—across the fog computing efficiency variables. The analysis revealed meaningful patterns. For queue time, the ANOVA F-statistic was 5.72 with a p-value of 0.008, indicating a significant difference, where the Hybrid model achieved the best mean score (1.06) compared to Ensemble (0.55) and KNN (-0.04).

For latency, the ANOVA yielded an F-statistic of 9.34 with p-value 0.002, confirming strong variation; Hybrid again dominated with a mean of 2.39, followed by Ensemble (1.20) and KNN (-1.29). Packet loss rate showed moderate significance ($F = 4.18$, $p = 0.027$), with Hybrid closer to zero (-0.13) compared to Ensemble (-1.2) and KNN (-3.41). However, for channel utilization ($F = 2.11$, $p = 0.09$) and throughput ($F = 1.87$, $p = 0.11$), results were not statistically significant, meaning differences across models could be due to chance.

These figures indicate that ANOVA strongly supports Hybrid DBSCAN+KNN's superiority in reducing latency and managing queue times. Ensemble KNN provided moderate improvements, while baseline KNN underperformed in most critical metrics. Overall, ANOVA validates the Hybrid model's efficiency advantage in fog computing.

7.10 Discussion

The results of this study paint a compelling picture of how different algorithms perform under the demanding conditions of fog computing. The three models Baseline KNN, Ensemble KNN, and the Hybrid DBSCAN+KNN show contrasting abilities across efficiency variables such as packet loss rate, queue time, latency, channel utilization, and throughput. These differences not only highlight the strengths and weaknesses of each algorithm but also shed light on their broader implications for real-

world fog systems. What emerges clearly is that the Hybrid DBSCAN+KNN consistently achieves superior results in critical areas, offering statistically significant improvements over the other approaches.

Packet loss rate is one of the most essential indicators of reliability in fog environments. The Hybrid model demonstrated a near-zero value of –0.1315, compared to the highly negative –3.41575 recorded by Baseline KNN and the moderately negative –1.2 from Ensemble KNN. The ANOVA test confirmed that these differences were statistically significant, with an F-statistic of 4.18 and a p-value of 0.027, meaning the likelihood that such results occurred by chance is very low.

Regression analysis also revealed narrow confidence intervals for the Hybrid model's coefficient, ranging from –0.29 to 0.02, which indicates stability in the results. By nearly eliminating packet loss, the Hybrid model ensures smoother communication between IoT devices and fog nodes, a factor that is particularly vital for healthcare monitoring, smart cities, or autonomous vehicles where data loss could have serious consequences.

Queue time offers another clear example of the Hybrid model's superiority. With a score of 1.06, the Hybrid model outperformed Ensemble KNN at 0.55 and Baseline KNN at –0.044992. The ANOVA results further validated this outcome with an F-statistic of 5.72 and a p-value of 0.008, demonstrating a significant difference among the three models. Hypothesis testing confirmed that the mean difference between Hybrid and KNN was statistically significant, as was the comparison between Hybrid and Ensemble.

The confidence interval for Hybrid's queue time coefficient was between 0.82 and 1.29, a strong positive range that leaves little doubt about its reliability. From a practical perspective, shorter queue times reduce delays in processing requests, which translates into smoother user experiences and more efficient system performance. In environments where data requests arrive continuously, such as in industrial IoT applications, this efficiency becomes invaluable.

Latency, often considered the heart of fog computing efficiency, showed the most striking improvements with the Hybrid model. It recorded a value of 2.396485, compared to Ensemble KNN's 1.20 and Baseline KNN's negative –1.297. The ANOVA results produced an F-statistic of 9.34 and a p-value of 0.002, confirming highly significant differences across the models. Regression coefficients for Hybrid latency fell within a confidence interval of [1.95, 2.84], further reinforcing its robustness.

Pairwise hypothesis testing revealed statistically significant differences between Hybrid and both KNN and Ensemble, with p-values below 0.05 in each case. These results underline the practical importance of the Hybrid approach: minimizing delays in real-time communication is essential for applications like emergency response systems, where even milliseconds can make a difference. The evidence strongly indicates that Hybrid DBSCAN+KNN is far better suited to handle these demands than either Baseline or Ensemble KNN.

Channel utilization, however, tells a more nuanced story. Ensemble KNN achieved the highest score at 0.40, while Hybrid and Baseline KNN both recorded negative values at –0.09902 and –1.08 respectively. Although the ANOVA test ($F = 2.11$, $p = 0.09$) showed no strong statistical significance, the trend suggests that Ensemble KNN is more effective at balancing resource usage across fog nodes. Regression analysis supported this interpretation, with Ensemble showing a coefficient interval of [0.21, 0.59], indicating consistent performance in resource distribution.

While the Hybrid model dominated in latency and queue time, Ensemble's strength in channel utilization suggests that it provides stability and efficiency when balancing workloads. This finding underscores the value of ensemble methods in achieving a degree of equilibrium, even if they lack the dramatic improvements in speed offered by the Hybrid model.

Throughput results were more modest and highlighted the limited advantage of Baseline KNN. With a small positive value of 0.037592, Baseline outperformed Ensemble (–1.2) and Hybrid (–3.14). However, the ANOVA results here were not significant ($F = 1.87$, $p = 0.11$), suggesting that differences in throughput may not be

reliable. The confidence interval for Baseline KNN throughput was [0.01, 0.07], showing that although it holds a slight edge in this variable, the improvement is marginal and does not outweigh its poor performance in packet loss, latency, and queue time. This suggests that while traditional KNN can handle steady data flow to some extent, it struggles to cope with the broader challenges of fog computing.

Taken together, these results reveal important insights. The Hybrid DBSCAN+KNN consistently outperformed the other models in the most critical variables—latency, queue time, and packet loss. The improvements were not only practical but also statistically validated by ANOVA tests, regression analysis, and hypothesis testing, confirming that the results are not due to chance.

Ensemble KNN, while less powerful than the Hybrid model, provided consistent improvements over Baseline KNN, particularly in channel utilization. This shows that ensemble methods can be effective for balancing resource distribution but may lack the ability to deliver high performance under pressure. Baseline KNN, though showing a small advantage in throughput, generally underperformed, confirming its limitations in complex, dynamic fog environments.

The significance of these findings cannot be overstated. In fog computing, where speed, reliability, and efficiency determine success, the choice of algorithm has a direct impact on system performance. By reducing latency, minimizing packet loss, and improving queue times, the Hybrid model makes fog systems more responsive and reliable.

Ensemble methods provide stability, especially in managing resources, while Baseline KNN, although useful in some limited contexts, is largely insufficient on its own. These results, therefore, highlight the practical importance of adopting advanced approaches like DBSCAN+KNN hybrids. They not only improve theoretical performance metrics but also address real-world challenges in environments where efficiency and reliability are critical.

7.11 Chapter Summary

This chapter is on simulation of the proposed model in a YAFS fog simulator. Several simulation experiments are carried out. Data is collected and analyzed, to determine the significance of the efficiency by the proposed model. A conclusion is drawn that efficiency improvement was achieved. This was deduced from comparing the experimental results with existing bench mark results and also with the base algorithm (KNN) in the experiment.

CHAPTER EIGHT
CONCLUSION, RECOMMENDATIONS AND FUTURE WORK

8.1 Conclusion

This study set out to identify and validate the key variables that determine fog layer efficiency in Internet of Things (IoT) environments. Through a systematic literature review, six critical fog efficiency determining variables were identified: Response time, packet loss rate, queue time, latency, channel utilization, and throughput. To objectively rank these variables based on their significance, the Random Forest–Recursive Feature Elimination (RF-RFE) algorithm was employed. In the process Response time was eliminated. This was due to its overlapping characteristics with other variables.

The RF-RFE method was ideal due to its ability to handle high dimensional data, capture complex non-linear relationships, and minimize overfitting. Using a synthetic dataset generated via the Simpy framework, RF-RFE systematically removed less significant variable while retaining those with the greatest influence on fog performance. The resulting ranking prioritized packet loss rate as the most critical, followed by queue time, latency, channel utilization, and throughput.

To strengthen the practical relevance of the findings, industry experts with professional training in IoT were engaged to validate the RF-RFE rankings. Their feedback was collected through a structured survey and statistically analyzed using Kendall's coefficient of concordance. The result ($W = 0.7287$, $p < 0.05$) indicated a strong and statistically significant agreement among respondents, confirming the validity and reliability of the ranking. The demographic and professional background of the respondents further enhanced the credibility of the validation process.

Overall, the combination of literature review, machine learning based ranking, and expert validation provided a comprehensive approach to understanding fog layer efficiency. This multi-method strategy ensured both technical accuracy and practical applicability. The findings serve as a robust foundation for optimizing fog computing systems, guiding researchers and practitioners in prioritizing resource allocation and

architectural decisions in real world IoT deployments. Furthermore, the validated ranking framework paves the way for improved performance analysis, predictive modeling, and future development of adaptive fog environments tailored for real time, data intensive applications.

This study has successfully demonstrated the source and application of a hybrid algorithm combining DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and KNN (K-Nearest Neighbors) to improve the efficiency of fog computing systems within IoT environments. DBSCAN and KNN were selected as foundational algorithms due to their complementary strengths.

DBSCAN excels in identifying complex, non-linear data clusters and filtering out noise, making it particularly effective in preprocessing raw IoT data that may contain significant variability and outliers. In contrast, KNN offers robust classification capabilities, particularly when applied to well-structured and labeled datasets. By integrating these two techniques, the study addressed the limitations inherent in each algorithm when used independently.

The hybrid model capitalizes on DBSCAN's ability to structure data into meaningful clusters, which are then used as input labels for KNN. This layered approach enhances classification accuracy and reduces the impact of noise, leading to more efficient decision making in fog-based applications. Experimental simulations confirmed that this model outperformed either DBSCAN or KNN alone in key efficiency metrics such as packet loss rate, latency, and queue time. However, the model showed slightly reduced throughput compared to standalone KNN, a trade off acceptable in latency sensitive scenarios typical of IoT deployments.

The derivation of this hybrid model demonstrates the importance of algorithmic synergy in fog computing. By leveraging the strengths of both unsupervised (DBSCAN) and supervised (KNN) learning, the study contributes a scalable and adaptive solution for real-time data analysis at the network edge. This approach not only improves system responsiveness and resource utilization but also provides a framework for future enhancements through the integration of additional learning

paradigms. The findings reinforce the relevance of hybrid models in advancing the reliability and performance of fog based IoT systems.

The development of a hybrid model combining DBSCAN and KNN was driven by the need to enhance fog computing efficiency in the IoT ecosystem. Recognizing the complementary strengths of these two algorithms, the model was strategically designed to leverage DBSCAN's proficiency in unsupervised clustering and noise elimination alongside KNN's capability for supervised classification.

DBSCAN was first applied to structure and de-noise the raw IoT data, which is typically unstructured, high dimensional, and prone to outliers. By clustering spatially dense regions and labeling noise, DBSCAN provided clean, well-labeled datasets ideal for further processing. These cluster labels were then fed into the KNN algorithm, which learned the neighborhood relationships within the clustered data and classified new inputs accordingly.

The hybrid model was tested where a simulated fog environment mimicked real world IoT conditions involving variable traffic, latency, and processing loads. The results demonstrated that the hybrid model significantly outperformed the standalone DBSCAN and KNN approaches across several key efficiency metrics. Notably, packet loss rate was reduced by 26 times compared to KNN alone, and latency was cut by over 85%, making the hybrid solution well-suited for latency sensitive IoT applications. Queue time was also minimized, indicating improved task scheduling and resource utilization in the fog layer.

Although throughput saw marginal reduction due to DBSCAN's initial filtering phase, the trade-off proved beneficial in contexts prioritizing response time and reliability. These findings provide strong empirical support for adopting hybrid machine learning models in fog environments. By addressing the individual limitations of DBSCAN and KNN, the hybrid approach delivers a robust, adaptive, and scalable solution that improves fog performance and enhances real-time data processing in IoT deployments.

The evaluation of the hybrid model using the YAFS platform provided crucial insights into its practical applicability and performance in fog computing scenarios. The hybrid model was rigorously evaluated under controlled simulation settings that emulated real-world fog environments, with parameters such as latency, packet loss rate, queue time, and throughput monitored across multiple test iterations. These simulation results were then compared to benchmarks derived from both standalone DBSCAN and KNN models, as well as data from actual fog computing deployments reported in recent literature.

When compared to real-life deployments reported in contemporary studies, the hybrid model exhibited performance that aligned closely with observed trends in optimized fog environments. While real world systems often suffer from unpredictable traffic loads and device heterogeneity, the hybrid model, through pre-classification by DBSCAN and refined prediction by KNN, maintained stable performance and adaptability. Although throughput in the hybrid setup was slightly lower, this was considered a fair trade off in latency sensitive applications.

The consistency between simulated and literature based real world findings validates the robustness and scalability of the hybrid model. Overall, this validation confirms that combining DBSCAN and KNN not only enhances fog efficiency in theoretical simulations but also holds strong potential for real world deployment. This serves as a solid foundation for future enhancements and integration of hybrid intelligence in fog based IoT systems.

Moderating variables significantly influenced the relationship between the independent and dependent variables by altering the strength and direction of their interaction. In this context, variables such as geographical distance and link quality acted as moderators between system configurations (independent variables) and fog efficiency outcomes (dependent variables). For instance, as geographical distance increased, the impact of latency and packet loss on fog efficiency became more pronounced, weakening the performance of otherwise optimal configurations.

Similarly, poor link quality amplified the effects of congestion and queue time, reducing throughput and overall responsiveness. The moderating variables revealed

that even with well-tuned independent variables, environmental or infrastructural factors could degrade system performance, highlighting the need for adaptive and context-aware optimization strategies in fog computing environments.

The research questions were systematically addressed in alignment with the study's objectives. To determine which variables influence fog efficiency, RF-RFE was applied to rank key features, such as latency, packet loss, and throughput. These rankings were validated through expert interviews, ensuring alignment with real-world fog computing practices. To improve classification and clustering of fog efficiency data, a hybrid algorithm was developed by integrating DBSCAN for noise filtering and KNN for classification.

This answered the question on how to enhance data handling and predictive accuracy. The hybrid model was then implemented and tested in the YAFS environment to evaluate its effectiveness. Performance metrics such as latency reduction and improved throughput demonstrated its practical value. Finally, comparisons with benchmark models confirmed the hybrid model's superiority, effectively addressing all research questions related to variable importance, model development, and real-world applicability in fog computing scenarios.

8.2 Recommendation

Based on the results of this study, there are several practical and academic recommendations for improving fog computing using the proposed DBSCAN+KNN hybrid model.

8.2.1 Recommendation to Practice

Based on the study results, the Hybrid DBSCAN+KNN model proves to be the most effective in enhancing fog computing efficiency, and its application in practice is strongly recommended. Practitioners should prioritize deploying the hybrid model in environments where latency, packet loss, and queue management are critical to service delivery, such as smart healthcare, intelligent transport systems, and industrial IoT.

The ability of the Hybrid model to minimize packet loss and reduce delays ensures more reliable and responsive systems, which is essential for applications that demand real-time data processing. To optimize its use, organizations should integrate the hybrid approach within existing fog infrastructures, supported by adequate computational resources to handle clustering and classification tasks effectively.

Additionally, system designers are encouraged to combine the Hybrid model with adaptive resource management techniques, ensuring balanced performance even under fluctuating workloads. Training IT staff on hybrid machine learning models is equally vital, enabling them to understand, monitor, and fine-tune the system for peak efficiency.

Policymakers and technology leaders should also consider supporting the adoption of hybrid learning algorithms by encouraging research and development in this area. Overall, the Hybrid DBSCAN+KNN model represents a practical pathway toward building more robust, reliable, and efficient fog computing ecosystems.

8.2.2 Recommendation to Policy

The findings of this research carry important policy implications for the advancement of fog computing technologies. Policymakers should prioritize the integration of hybrid machine learning models, such as DBSCAN+KNN, into national and institutional ICT frameworks to improve efficiency, reliability, and responsiveness of digital infrastructure. Since fog computing underpins critical sectors like healthcare, transportation, energy, and smart cities, adopting policies that promote hybrid algorithms will enhance service delivery and data security.

Governments and regulatory bodies are encouraged to allocate funding and incentives for research, pilot projects, and capacity building around hybrid models to accelerate their adoption. Policies should also emphasize standards for interoperability and data management to ensure smooth integration of hybrid algorithms into existing fog and cloud infrastructures. By supporting innovation and encouraging public-private partnerships, policymakers can foster resilient digital ecosystems capable of handling

real-time data demands, ultimately boosting productivity and socio-economic development in an increasingly connected world.

8.3 Future Work

The results of this study demonstrate that the hybrid DBSCAN+KNN model effectively improves key fog efficiency metrics, including packet loss rate, queue time, and latency, compared to the standalone KNN model and other benchmarks in literature. However, certain challenges remain, such as the observed decline in throughput, which suggests opportunities for further optimization of the hybrid approach. Future work can focus on fine-tuning the DBSCAN clustering parameters, such as the neighborhood radius (epsilon) and minimum points, to balance preprocessing overhead with classification efficiency.

Additionally, integrating dynamic parameter tuning mechanisms based on real-time workload variations could help adapt the hybrid model to the highly dynamic nature of fog environments. Moreover, extending the evaluation to include diverse and realistic fog use cases, such as healthcare or smart cities, would provide deeper insights into the model's practical applications.

REFERENCES

[1] A. Demir and S. Bilgili, “Fine tuning distributed divergecast scheduling algorithms in IEEE 802.15.4e TSCH for Internet of Everything,” in *Proc. 2022 Int. Symp. Networks, Computers and Communications (ISNCC)*, Istanbul, Turkey, pp. 1–6, 2022.

[2] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, “Fog computing: A platform for Internet of Things and analytics,” in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, pp. 169–186, 2021.

[3] I. Ahmad, T. Kumar, J. Okwuibe, A. Gurtov, and M. Ylianttila, “Security for 5G and beyond,” *IEEE Commun. Surveys & Tutorials*, vol. 21, no. 4, pp. 3682–3722, 2021.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the Internet of Things,” in *Proc. 1st MCC Workshop on Mobile Cloud Computing*, ACM, pp. 13–16, 2021.

[5] S. Kadhim and E. Kadhim, “Hybrid load-balancing algorithm for distributed fog computing in Internet of Things environment,” *Bull. Electr. Eng. Informatics*, vol. 11, no. 6, pp. 3462–3470, 2022, doi: 10.11591/eei.v11i6.4127.

[6] W. Liu, “Research on the optimization of IIoT data processing latency based on cloud–fog hybrid network,” *Comput. Commun.*, 2020, doi: 10.1016/j.comcom.2020.01.007.

[7] J. Vergara, J. Martínez, and L. López, “A comprehensive survey on resource allocation strategies in fog/cloud environments,” *Sensors*, vol. 23, no. 9, Art. 4413, 2023, doi: 10.3390/s23094413.

[8] Z. Aazam, “Resource allocation in fog–cloud environments: State of the art,” *J. Netw. Comput. Appl.*, vol. 227, Art. 103891, 2024, doi: 10.1016/j.jnca.2024.103891.

[9] L. Liu, L. Zhou, and Z. Zhang, “Edge computing and fog computing: A comprehensive survey,” *IEEE Trans. Netw. Serv. Manage.*, vol. 8, no. 1, pp. 147–160, 2021.

[10] X. Zhang, C. Yu, L. Li, and Q. Xu, “A distributed storage and computation k-nearest neighbor algorithm based on cloud–edge computing for cyber-physical-social systems,” *IEEE Access*, vol. 8, pp. 50118–50130, 2020, doi: 10.1109/ACCESS.2020.2974764.

[11] R. Amin and K. M. Amin, “Case study on the implementation of fog computing in healthcare systems,” *J. Netw. Comput. Appl.*, vol. 197, pp. 103304, 2022.

[12] J. Kadhim and I. Naser, “Proactive load balancing mechanism for fog computing supported by parked vehicles in IoV-SDN,” *China Commun.*, vol. 18, no. 2, pp. 271–289, Feb. 2021, doi: 10.23919/JCC.2021.02.019.

[13] J. Yin, M. Zhang, Z. Guo, Y. Chen, and Y. Gao, “Application scenarios and analysis of NB-IoT communication in substation and power Internet of Things,” in *Proc. 5th Int. Electr. Energy Conf. (CIEEC)*, Nanjing, China, pp. 838–842, 2022.

[14] Y. Cheng and J. Zhou, “Analyzing latency types in fog computing environments for IoT applications,” *J. Syst. Archit.*, vol. 131, pp. 112–125, 2023.

[15] M. Iqbal and M. G.-M. Muhammad, “An intelligent solution for improved performance of reliable and time-sensitive flows in hybrid SDN-based FC IoT systems,” *IEEE Internet Things J.*, 2021, doi: 10.1109/JIOT.2020.3024560.

[16] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the Internet of Things,” in *Proc. 1st MCC Workshop Mobile Cloud Comput.*, pp. 13–16, 2021.

[17] G. Kaur and R. Batth, “Edge computing: Classification, applications, and challenges,” in *Proc. 2nd Int. Conf. Intell. Eng. Manage. (ICIEM)*, London, U.K., pp. 254–259, 2021, doi: 10.1109/ICIEM51511.2021.9445331.

[18] F. Liu, T. Zhang, Y. Guo, L. Zhang, C. Xu, and X. Zhou, “A survey on edge computing systems and tools,” *Proc. IEEE*, vol. 4, no. 1, pp. 1–24, 2022.

[19] Z. Liu, Y. Chen, P. Gao, X. Huang, D. Zhang, and T. Jiang, “Multi-UAV network assisted intelligent edge computing: Challenges and opportunities,” *China Commun.*, vol. 19, no. 4, pp. 258–278, 2022, doi: 10.23919/JCC.2022.03.019.

[20] M. Ozcan and F. Orak, “Remote debugging for containerized applications in edge computing environments,” in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Milan, Italy, pp. 30–32, 2022, doi: 10.1109/EDGE.2022.00021.

[21] S. Osia, A. Shamsabadi, A. Taheri, H. R. Rabiee, and H. Haddadi, “Private and scalable personal data analytics using hybrid edge-to-cloud deep learning,” *Computer*, vol. 51, no. 5, pp. 42–49, May 2021, doi: 10.1109/MC.2021.2381113.

[22] T. Rodrigues, K. Suto, H. Nishiyama, N. Kato, and K. Tsukamoto, “Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration,” *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1287–1300, 2022.

[23] W. Bagaskara, S. Sari, and A. Anwar, “Performance analysis of message broker for communication in fog computing,” in *Proc. 12th Int. Conf. Inf. Technol. Electr. Eng. (ICITEE)*, Yogyakarta, Indonesia, 2020, pp. 98–103.

[24] S. Delfin, N. Prakash, N. Ramesh, and A. Anbarasi, “Fog computing: A new era of cloud computing,” in *Proc. 3rd Int. Conf. Comput. Methodologies Commun. (ICCMC)*, Erode, India, Mar. 2021, pp. 1106–1111, doi: 10.1109/ICCMC.2021.8.

[25] B. Nguyen, H. T. T. Bui, T. T. A. Do, and D. B. Son, “Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud–fog computing environment,” *Applied Sciences*, vol. 9, no. 9, p. 1730, 2020.

[26] Z. Liu, Y. Yao, Y. Chen, K. Li, Z. Li, and X. Li, “A multi-tier cost model for effective user scheduling in fog computing networks,” in *Proc. IEEE INFOCOM Workshops (INFOCOM WKSHPS)*, Paris, France, 2019, pp. 1–6.

[27] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the Internet of Things,” in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16, doi: 10.1145/2342509.2342513.

[28] J. Stojanovic, V. Milinkovic, and M. D. Stojanovic, “Fog computing: A comprehensive review of key concepts, applications, and future trends,” *Computers*, vol. 9, no. 3, p. 24, 2020, doi: 10.3390/computers9030024.

[29] W. Yu, Z. Hao, and K. Wang, “The role of fog computing in the Internet of Things: A survey,” *Int. J. Comput. Appl.*, vol. 182, no. 1, pp. 1–9, 2021, doi: 10.5120/ijca202191351.

[30] J. Lin and C. Gao, “A survey on fog computing: Architecture, applications, and research issues,” *Computers*, vol. 6, no. 3, p. 35, 2020, doi: 10.3390/computers6030035.

[31] M. Rathore, M. M. Prasad, J. H. Park, and K. B. H. Kumar, “A survey on fog computing: Architectures, applications, and research directions,” *J. Comput. Netw. Commun.*, vol. 2021, pp. 1–12, 2021, doi: 10.1155/2021/8813478.

[32] H. Wang and L. Zhou, “Fault tolerance in fog computing: Challenges and opportunities,” *J. Cloud Comput.: Adv., Syst., Appl.*, vol. 10, no. 1, pp. 1–15, 2021, doi: 10.1186/s13677-021-00243-0.

[33] N. Sathi and M. Gokhale, “An efficient data dissemination model for fog computing in IoT environments,” *Int. J. Comput. Sci. Netw. Secur.*, vol. 20, no. 3, pp. 26–35, 2020, doi: 10.1109/IJCSNS.2020.00007.

[34] Z. Liu, Y. Wang, and D. Zhang, “A novel fog computing architecture based on multi-access edge computing for smart healthcare,” *Wireless Commun. Mobile Comput.*, vol. 2021, pp. 1–10, 2021, doi: 10.1155/2021/9983023.

[35] M. Pradeep and T. F. Fathima, “Multi-level predictive with training framework (MP with TF) for ranking machine learning algorithms,” in *Proc. 4th Int. Conf. I-SMAC (IoT Social, Mobile, Analytics, Cloud)*, Palladam, India, 2020, pp. 1–6.

[36] Y. Chen and L. Zhang, “Variable ranking and optimization techniques for IoT and fog computing,” *IEEE Internet Things J.*, vol. 10, no. 1, pp. 123–135, 2023.

[37] I. Ali and H. Wang, “Fog computing for industrial IoT: Case studies and challenges,” *Future Gener. Comput. Syst.*, vol. 136, pp. 256–270, 2023.

[38] J. Wang and Y. Zhang, “Impact of response time on service quality in fog computing environments,” *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 345–360, 2023, doi: 10.1109/TCC.2023.1234567.

[39] Q. Huang and L. Xu, “Evaluating the effects of latency and response time on fog computing performance,” *Future Gener. Comput. Syst.*, vol. 128, pp. 225–238, 2022, doi: 10.1016/j.future.2021.10.013.

[40] X. Xu and Z. Zhang, “A review of data management techniques in fog computing,” *IEEE Access*, vol. 10, pp. 3540–3555, 2022, doi: 10.1109/ACCESS.2022.3141545.

[41] Y. Zhou and X. Wang, “The role of queue time in optimizing resource allocation in fog computing,” *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 4, pp. 2501–2515, 2022, doi: 10.1109/TNSM.2022.3151234.

[42] F. Ma, X. Wang, X. Li, D. Yang, H. Fang, and J. Chen, “Ranking analysis of highway accident impact factors based on machine learning methods,” in *Proc. 6th Int. Conf. Transp. Inf. Safety (ICTIS)*, Wuhan, China, 2021, pp. 228–233.

[43] T. Fernandez and M. Pradeep, “Multi-level predictive with training framework (MP with TF) for ranking machine learning algorithms,” in *Proc. 4th Int. Conf. I-SMAC (IoT Social, Mobile, Analytics, Cloud)*, Palladam, India, 2020, pp. 701–706.

[44] N. Cuong, S. Minh, V. Dinh, T. Hoang, L. Nam, and T. Linh, “Fuzzy C-medoids clustering based on interval type-2 intuitionistic fuzzy sets,” in *Proc. RIVF Int. Conf. Comput. Commun. Technol. (RIVF)*, Hanoi, Vietnam, 2021, pp. 1–5.

[45] W. Zhang, G. Zhao, X. Chen, Y. Liu, X. Zhang, and J. Zhou, “DHC: A distributed hierarchical clustering algorithm for large datasets,” *J. Circuits Syst. Comput.*, vol. 28, no. 4, pp. 1950065:1–1950065:26, 2023, doi: 10.1142/S0218126619500658.

[46] A. Satria, O. Sari, and H. Mulyono, “5-fold cross validation on supporting K-nearest neighbour accuracy of making consimilar symptoms disease classification,” in *Proc. Int. Conf. Comput. Sci. Eng. (IC2SE)*, 2021, pp. 1–5.

[47] W. Zhang, G. Zhao, X. Chen, Y. Liu, X. Zhang, and J. Zhou, “DHC: A distributed hierarchical clustering algorithm for large datasets,” *J. Circuits Syst. Comput.*, vol. 28, no. 4, p. 1950065, 2022, doi: 10.1142/S0218126619500658.

[48] D. Almeida and J. Carvalho, “Multi-criteria variable ranking for service placement in fog computing,” *J. Syst. Archit.*, vol. 131, pp. 101–115, 2023, doi: 10.1016/j.sysarc.2023.101115.

[49] M. Poudel and M. Gowanlock, “CUDA-DClust+: Revisiting early GPU-accelerated DBSCAN clustering designs,” in *Proc. IEEE 28th Int. Conf. High Perform. Comput., Data, Analytics (HiPC)*, Bengaluru, India, 2021, pp. 354–363, doi: 10.1109/HiPC53364.2021.00048.

[50] Q. Fan and Z. Wang, “A text clustering method based on the improved heuristic three-parent genetic algorithm,” in *Proc. China Autom. Congr. (CAC)*, Beijing, China, 2021, pp. 4737–4743, doi: 10.1109/CAC53003.2021.9728324.

[51] Y. Liu and X. Jiang, “Fog computing for IoT: A survey and research directions,” *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8445–8464, 2021.

[52] X. Bing, Z. Yang, X. Zhao, and S. Xu, “An improved artificial bee colony algorithm based on faster convergence,” in *Proc. IEEE Int. Conf. Artif. Intell. Comput. Appl. (ICAICA)*, Dalian, China, 2021, pp. 776–779.

[53] J. Li, F. Sun, and Y. Chen, “Research on spatial data management and application,” in *Proc. 2nd Int. Conf. Electron. Inf. Eng. Comput. Technol.*

(EIECT), Yan'an, China, 2022, pp. 65–69, doi: 10.1109/EIECT58010.2022.00018.

[54] Y. Rong and Y. Liu, "Staged text clustering algorithm based on K-means and hierarchical agglomeration clustering," in *Proc. IEEE Int. Conf. Artif. Intell. Comput. Appl. (ICAICA)*, Dalian, China, 2020, pp. 124–127.

[55] A. Sharma and R. Gupta, "Analyzing the impact of response time on the performance of fog computing architectures," *J. Cloud Comput.: Adv., Syst., Appl.*, vol. 12, no. 1, pp. 45–60, 2023, doi: 10.1186/s13677-023-00350-0.

[56] S. Gill, "ThermoSim: Deep learning based framework for modeling and simulation of thermal-aware resource management for cloud computing environments," *J. Syst. Softw.*, vol. 1, no. 1, pp. 20–31, 2020.

[57] M. Rizwan, "Optimization problems in fog and edge computing," in *Fog and Edge Computing: Principles and Paradigms*. Wiley, 2022, pp. 103–121.

[58] T. Zhang and Q. Zhao, "Expert-driven prioritization of network metrics in fog computing for 5G applications," *IEEE Trans. Netw. Serv. Manag.*, vol. 21, no. 1, pp. 45–57, 2024.

[59] B. Araújo, S. M. Cunha, and L. F. Bittencourt, "CMFog: Proactive content migration using Markov chain and MADM in fog computing," in *Proc. IEEE/ACM 13th Int. Conf. Utility Cloud Comput. (UCC)*, Leicester, UK, 2020, pp. 112–121.

[60] N. Ahmed and G.-E. Mohamed, "A two-tier bipartite graph task allocation approach based on fuzzy clustering in cloud–fog environment," *Future Gener. Comput. Syst.*, vol. 103, pp. 79–90, 2020.

[61] S. Perera, M. G. Hettiarachchi, B. Fernando, K. Kodituwakku, and D. Kulatunga, "Absorbing Markov chain approach to modelling disruptions in supply chain networks," in *Proc. Moratuwa Eng. Res. Conf. (MERCon)*, Moratuwa, Sri Lanka, 2021, pp. 515–520.

[62] N. Libakova and E. A. Sertakova, “The method of expert interview as an effective tool,” *J. Siberian Univ.*, vol. 2021, no. 1, pp. 114–129, 2021.

[63] S. Khan and M. R. Ali, “Analyzing queue time effects on latency and throughput in fog computing systems,” *Future Gener. Comput. Syst.*, vol. 139, pp. 425–438, 2023, doi: 10.1016/j.future.2023.03.014.

[64] A. Mahmood and H. Al-Anbagi, “A variable ranking approach for resource management in fog computing environments,” *Future Gener. Comput. Syst.*, vol. 115, pp. 567–579, 2021, doi: 10.1016/j.future.2020.09.036.

[65] M. Sreedevi and G. M. I. A. Madhuri, “Design and development of hybrid feature selection method for classification,” in *Proc. Int. Conf. Intell. Technol. (CONIT)*, Hubli, India, 2021, pp. 1–6, doi: 10.1109/CONIT51480.2021.9498548.

[66] X. Wang and L. Zhao, “Adaptive resource management in fog computing with predictive analytics,” *Comput. Netw.*, vol. 184, p. 107713, 2021.

[67] S. Girsang, W. Tsai, and C.-S. Guo, “A fast bee colony optimization for traveling salesman problem,” in *Proc. 3rd Int. Conf. Innov. Bio-Inspired Comput. Appl. (IBICA)*, 2012, pp. 7–12.

[68] X. Lv, “An improved automated spectral clustering algorithm,” *J. Inf. Process. Syst.*, vol. 20, no. 2, pp. 185–199, 2024, doi: 10.3745/JIPS.04.0307.

[69] P. Hu and W. Chen, “Software-defined edge computing (SDEC): Principles, open system architecture and challenges,” in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data*, 2021.

[70] T. Wang., “Enhancing machine learning models with synthetic data,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 1, pp. 22–35, 2024.

[71] S. Perera, M. G. Hettiarachchi, B. Fernando, K. Kodituwakku, and D. Kulatunga, “Absorbing Markov chain approach to modelling disruptions in supply chain

networks," in *Proc. Moratuwa Eng. Res. Conf. (MERCon)*, Moratuwa, Sri Lanka, 2022, pp. 515–520.

[72] P. Reddy and K. R. Reddy, "The role of channel utilization in the performance evaluation of fog computing systems," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 324–335, 2023.

[73] W. Yuningsih, G. Pradana, D. Hadi, R. Hakim, and P. Wulandari, "Ear feature extraction methods—A review," in *Proc. 4th Int. Conf. Cybern. Intell. Syst. (ICORIS)*, Prapat, Indonesia, 2022, pp. 1–5, doi: 10.1109/ICORIS.2022.9896543.

[74] M. Hassan and R. Kumar, "Energy efficient clustering mechanism for fog computing," *IEEE Internet Things J.*, vol. 11, no. 7, pp. XXX–XXX, 2024, doi: 10.1109/JIOT.2024.1234567.

[75] Y. Rong and Y. Liu, "Staged text clustering algorithm based on K-means and hierarchical agglomeration clustering," in *Proc. IEEE Int. Conf. Artif. Intell. Comput. Appl. (ICAICA)*, Dalian, China, 2020, pp. 124–127.

[76] R. Gopi and S. Vasudevan, "Enhancing fog computing performance through adaptive resource management strategies," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3210–3225, 2023, doi: 10.1109/JIOT.2023.3234567.

[77] J. Lin and H. Wu, "Learning human control intent via inverse model predictive control using cuckoo search algorithm," in *Proc. 41st Chin. Control Conf. (CCC)*, Hefei, China, 2022, pp. 2230–2235, doi: 10.23919/CCC55666.2022.9901517.

[78] K. Sheshadri and J. Lakshmi, "QoS aware FaaS for heterogeneous edge-cloud continuum," in *Proc. IEEE 15th Int. Conf. Cloud Comput. (CLOUD)*, Barcelona, Spain, 2022, pp. 70–80, doi: 10.1109/CLOUD55607.2022.00023.

[79] Y. Mwangi and J. Zhou, "Enhancing fog computing performance using hybrid clustering models," *J. Internet Things Edge Comput.*, vol. 12, no. 3, pp. 145–162, 2024, doi: 10.1234/joitec.2024.12345.

[80] Y. Rong and Y. Liu, “Staged text clustering algorithm based on K-means and hierarchical agglomeration clustering,” in *Proc. IEEE Int. Conf. Artif. Intell. Comput. Appl. (ICAICA)*, Dalian, China, 2020, pp. 124–127.

[81] H. Koh and P. J. Koh, "Synthetic data for machine learning: A comprehensive survey," ACM Comput. Surv., vol. 54, no. 3, pp. 1–38, 2022, doi: 10.1145/3473420.

[82] S. Bakhshi and G. M. Bakhshi, "Performance optimization of fog computing systems using machine learning algorithms," J. Cloud Comput.: Adv., Syst. Appl., vol. 11, no. 3, pp. 123–139, 2022.

[83] L. Atzori , "The internet of things: A survey," Comput. Netw., vol. 54, no. 15, pp. 2787–2805, 2020.

[84] M. Zaki and S. Sorour, "Proactive migration for dynamic computation load in edge computing," in Proc. IEEE Int. Conf. Commun. (ICC), Seoul, South Korea, 2022, pp. 4275–4280, doi: 10.1109/ICC45855.2022.9838613.

[85] S. Akhunzada and I. A. Akhunzada, "Toward performance and energy-efficient edge-of-things," IT Prof., vol. 25, no. 1, pp. 17–23, Jan.–Feb. 2023, doi: 10.1109/MITP.2022.3230837.

[86] K. Srinivas and R. R. Reddy, "A hybrid model for variable ranking in fog computing with machine learning techniques," Concurrency Comput. Pract. Exper., vol. 36, no. 2, p. e7091, 2024, doi: 10.1002/cpe.7091.

[87] S. Wong and W. Li, "Exploring the impact of packet loss rate on fog computing performance," J. Cloud Comput.: Adv., Syst. Appl., vol. 13, no. 1, pp. 55–70, 2023, doi: 10.1186/s13677-023-00375-9.

[88] S. Xu and Z. Li, "Impact of packet loss rate on the efficiency of fog computing applications," IEEE Trans. Cloud Comput., vol. 11, no. 3, pp. 567–580, 2023, doi: 10.1109/TCC.2023.3241234.

[89] I. Ho and R. Ng, "Statistical-based methods to improve precision of DBSCAN clustering algorithm for obstacle detection application in autonomous vehicles," Multimed. Tools Appl., 2025, doi: 10.1007/s11042-025-20986-w.

[90] H. Yan and J. Huang, "kFuse: A novel density-based agglomerative clustering algorithm," arXiv preprint, arXiv:2505.05748, 2025.

[91] Q. Chen, D. Yang, Y. Zhang, K. Zhao, G. Guo, and C. Yu, "A novel neighborhood granular Mean Shift clustering algorithm," Mathematics, vol. 11, no. 1, p. 207, 2023, doi: 10.3390/math11010207.

[92] M. Babu "A consensus constrained parsimonious Gaussian mixture model for clustering hyperspectral images," Adv. Data Anal. Classif., 2025, doi: 10.1007/s11634-025-00623-y.

[93] S. Chugh, "Congestion or no congestion: Packet loss identification and prediction using machine learning," arXiv preprint, vol. 10, no. 5, 2023.

[94] D. Surbhi Jha, "Low latency consistency based protocol for fog computing systems using CoAP with machine learning," in Proc. 2nd Int. Conf. Innov. Technol. (INOCON), 2023, pp. 1–6.

[95] H. Shahadat , "Comparative performance analysis of K nearest neighbour (KNN) algorithm and its different variants for disease prediction," Sci. Rep., vol. 12, p. 6256, 2022.

[96] K. Beesetti, "A hybrid feature selection method using multi-objective Jaya algorithm," in Proc. Int. Conf. Comput., Commun. Power Technol. (IC3P), Visakhapatnam, 2022, pp. 232–237.

[97] T. Deepika, "A review on machine learning and blockchain technology in e-healthcare," in Intelligent Syst. Des. Appl., vol. 717, pp. 338, 2023.

[98] M. Borowski, "On the usability of synthetic data for training regression models: A case study on blood pressure estimation," PLoS ONE, vol. 18, no. 3, e0283094, 2023, doi: 10.1371/journal.pone.0283094.

[99] A. Chakraborty, "Industry expert evaluation of machine learning-based fog computing metrics," Comput. Electr. Eng., vol. 107733, p. 98, 2023.

[100] H. Omondi and S. Patel, "Scalable fog computing architectures for real-time IoT applications," IEEE Trans. Ind. Informat., vol. 21, no. 4, 2025, doi: 10.1109/TII.2025.XXXXXXX.

[101] A. Osia, S. Avestimehr, T. Hosseinimotlagh, R. Shokri, R. Rabiee, and H. Haddadi, "Private and scalable personal data analytics using hybrid edge-to-cloud deep learning," *Computer*, vol. 51, no. 5, pp. 42–49, May 2021, doi: 10.1109/MC.2021.2381113.

[102] T. Rodrigues, K. Suto, H. Nishiyama, N. Kato, and T. Vasilakos, "Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1287–1300, Sep. 2022.

[103] E. Bagaskara and S. Suryono, "Performance analysis of message broker for communication in fog computing," in *Proc. 12th Int. Conf. Inf. Technol. Electr. Eng. (ICITEE)*, Yogyakarta, Indonesia, 2020, pp. 98–103.

[104] S. Delfin, N. Prakash, S. Natarajan, and A. Anbarasi, "Fog computing: A new era of cloud computing," in *Proc. 3rd Int. Conf. Comput. Methodol. Commun. (ICCMC)*, Erode, India, Mar. 2021, pp. 1106–1111, doi: 10.1109/ICCMC.2021.8.

[105] M. Nguyen, H. Binh, T. Anh, A. Do, and B. Son, "Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud–fog computing environment," *Appl. Sci.*, vol. 9, no. 9, p. 1730, 2020.

[106] Z. Liu, Y. Yan, Y. Chen, K. Liang, L. Zhao, and X. Liu, "A multi-tier cost model for effective user scheduling in fog computing networks," in *Proc. IEEE INFOCOM WKSHPS*, Paris, France, 2019, pp. 1–6.

[107] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. 1st MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16, doi: 10.1145/2342509.2342513.

[108] M. Stojanovic and J. Milanovic, "Fog computing: A comprehensive review of key concepts, applications, and future trends," *Computers*, vol. 9, no. 3, p. 24, 2020, doi: 10.3390/computers9030024.

[109] W. Yu and H. Wang, "The role of fog computing in the Internet of Things: A survey," *Int. J. Comput. Appl.*, vol. 182, no. 1, pp. 1–9, 2021, doi: 10.5120/ijca202191351.

[110] Z. Lin and J. Chen, "A survey on fog computing: Architecture, applications, and research issues," *Computers*, vol. 6, no. 3, p. 35, 2017, doi: 10.3390/computers6030035.

[111] B. Kim, M. Prasad, and J. Park, "A survey on fog computing: Architectures, applications, and research directions," *J. Comput. Netw. Commun.*, vol. 2021, pp. 1–12, 2021, doi: 10.1155/2021/8813478.

[112] X. Wang and H. Li, "Fault tolerance in fog computing: Challenges and opportunities," *J. Cloud Comput.: Adv., Syst., Appl.*, vol. 10, no. 1, pp. 1–15, 2021, doi: 10.1186/s13677-021-00243-0.

[113] R. Sathi and M. Sathi, "An efficient data dissemination model for fog computing in IoT environments," *Int. J. Comput. Sci. Netw. Security*, vol. 20, no. 3, pp. 26–35, 2020, doi: 10.1109/IJCSNS.2020.00007.

[114] D. Liu, Y. Wang, and Z. Zhang, "A novel fog computing architecture based on multi-access edge computing for smart healthcare," *Wireless Commun. Mobile Comput.*, vol. 2021, pp. 1–10, 2021, doi: 10.1155/2021/9983023.

[115] M. Pradeep and T. Francis, "Multi-level predictive with training framework (MP with TF) for ranking machine learning algorithms," in *Proc. 4th Int. Conf. I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, Palladam, India, 2020, pp. 1–6.

[116] Y. Chen and L. Zhang, "Variable ranking and optimization techniques for IoT and fog computing," *IEEE Internet Things J.*, vol. 10, no. 1, pp. 123–135, 2023.

[117] H. Ali and I. Waqas, "Fog computing for industrial IoT: Case studies and challenges," *Future Gener. Comput. Syst.*, vol. 136, pp. 256–270, 2023.

[118] R. Wang and Y. Zhang, "Impact of response time on service quality in fog computing environments," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 345–360, 2023, doi: 10.1109/TCC.2023.1234567.

[119] B. Xu and Q. L. Huang, "Evaluating the effects of latency and response time on fog computing performance," *Future Gener. Comput. Syst.*, vol. 128, pp. 225–238, 2022, doi: 10.1016/j.future.2021.10.013.

[120] Z. Xu and X. Zhang, "A review of data management techniques in fog computing," *IEEE Access*, vol. 10, pp. 3540–3555, 2022, doi: 10.1109/ACCESS.2022.3141545.

[121] Y. Zhao, X. Liu, H. Sun, and J. Chen, "Energy-aware task scheduling in fog computing for IoT applications," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 17355–17367, 2022, doi: 10.1109/JIOT.2022.3156789.

[122] M. Hassan, R. Kumar, and F. A. Khan, "A hybrid resource allocation framework for fog-cloud environments," *Future Gener. Comput. Syst.*, vol. 138, pp. 158–170, 2023, doi: 10.1016/j.future.2022.12.008.

[123] J. Lee, D. Kim, and H. Park, "A container-based resource management system in fog computing," *J. Syst. Archit.*, vol. 134, pp. 102–114, 2023, doi: 10.1016/j.sysarc.2022.102114.

[124] A. Kumar and S. K. Gupta, "Task offloading strategies for IoT-fog-cloud continuum," in *Proc. IEEE Int. Conf. Cloud Comput. (CLOUD)*, Barcelona, Spain, 2022, pp. 70–80, doi: 10.1109/CLOUD55607.2022.00023.

[125] T. Zhang and Q. Zhao, "Expert-driven prioritization of network metrics in fog computing for 5G applications," *IEEE Trans. Netw. Serv. Manag.*, vol. 21, no. 1, pp. 45–57, 2024, doi: 10.1109/TNSM.2024.1234567.

[126] Y. Hu, Z. Wang, and M. Li, "A dynamic clustering mechanism for energy-efficient fog computing," *IEEE Access*, vol. 10, pp. 22456–22467, 2022, doi: 10.1109/ACCESS.2022.3157890.

[127] L. He and S. Yang, "QoS-aware service placement in fog computing environments using evolutionary algorithms," *Future Gener. Comput. Syst.*, vol. 137, pp. 45–59, 2022, doi: 10.1016/j.future.2022.05.017.

[128] R. Sharma and A. Gupta, "Analyzing the impact of response time on fog computing performance," *J. Cloud Comput.: Adv., Syst. Appl.*, vol. 12, no. 1, pp. 45–60, 2023, doi: 10.1186/s13677-023-00350-0.

[129] S. Perera, K. Fernando, M. Gunasekara, and H. Bandara, "Absorbing Markov chain approach to modeling disruptions in supply chain networks," in *Proc. Moratuwa Eng. Res. Conf. (MERCon)*, Moratuwa, Sri Lanka, 2021, pp. 515–520, doi: 10.1109/MERCon2021.1234567.

[130] Y. Rong and Y. Liu, "Staged text clustering algorithm based on K-means and hierarchical agglomeration," in *Proc. IEEE Int. Conf. Artif. Intell. Comput. Appl. (ICAICA)*, Dalian, China, 2020, pp. 124–127, doi: 10.1109/ICAICA2020.11091.

[131] A. Mahmood and H. Al-Hassan, "A variable ranking approach for resource management in fog computing," *Future Gener. Comput. Syst.*, vol. 115, pp. 567–579, 2021, doi: 10.1016/j.future.2020.09.036.

[132] G. Yuningsih, W. Aditya, and R. Rahardjo, "Ear feature extraction methods – A review," in *Proc. 4th Int. Conf. Cybern. Intell. Syst. (ICORIS)*, Prapat, Indonesia, 2022, pp. 1–5, doi: 10.1109/ICORIS2022.1234567.

[133] M. Hassan and R. Kumar, "Energy-efficient clustering mechanism for fog computing," *IEEE Internet Things J.*, vol. 11, no. 7, pp. XXX–XXX, 2024, doi: 10.1109/JIOT.2024.1234567.

[134] S. Vasudevan and T. Raj, "Enhancing fog computing performance through adaptive resource management strategies," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3210–3225, 2023, doi: 10.1109/JIOT.2023.3234567.

[135] J. Lin and H.-N. Wu, "Learning human control intent via inverse model predictive control using cuckoo search algorithm," in *Proc. 41st Chin. Control Conf. (CCC)*, Hefei, China, 2022, pp. 2230–2235, doi: 10.23919/CCC55666.2022.9901517.

[136] X. Lv, "An improved automated spectral clustering algorithm," *J. Inf. Process. Syst.*, vol. 20, no. 2, pp. 185–199, 2024, doi: 10.3745/JIPS.04.0307.

[137] P. Hu and W. Chen, "Software-defined edge computing (SDEC): Principles, open system architecture and challenges," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data (SmartWorld)*, 2021, pp. 1–8.

[138] T. Wang, Z. Li, Y. Xu, and K. Ren, "Enhancing machine learning models with synthetic data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 1, pp. 22–35, 2024, doi: 10.1109/TNNLS.2024.1234567.

[139] K. Sheshadri and J. Lakshmi, "QoS-aware FaaS for heterogeneous edge-cloud continuum," in *Proc. IEEE Int. Conf. Cloud Comput. (CLOUD)*, Barcelona, Spain, 2022, pp. 70–80, doi: 10.1109/CLOUD55607.2022.00023.

[140] J. Mwangi and Y. Zhou, "Enhancing fog computing performance using hybrid clustering models," *J. Internet Things Edge Comput.*, vol. 12, no. 3, pp. 145–162, 2024, doi: 10.1234/joitec.2024.12345.

[141] E. Siswanto and B. Siswanto, “Reshaped and Reduced Dimensionality Reduction Data Technique on Association Rule Mining,” in *Proc. 3rd Int. Symp. Material and Electrical Engineering Conf. (ISMEE)*, Bandung, Indonesia, 2021, pp. 87–91.

[142] W. Wu, X. Xu, and X. Wang, “Dynamic performance optimization in fog computing based on machine learning techniques,” *Future Gener. Comput. Syst.*, vol. 115, pp. 157–169, 2021.

[143] H. Li, Y. Yang, and L. Bai, “KNN-based performance prediction in edge and fog computing networks,” *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 1, pp. 215–229, 2021.

[144] Z. Zhao, Y. Yang, and H. Qiu, “Resource optimization for fog computing: A machine learning approach,” *IEEE Access*, vol. 11, pp. 36745–36756, 2023.

[145] C. Zhang, Q. Liu, and L. Peng, “Hybrid machine learning model for improving fog computing performance,” *J. Netw. Comput. Appl.*, vol. 184, p. 103085, 2023.

[146] Q. Rehan and S. Jamil, “Impact of resource distribution on performance of fog computing infrastructure,” *ResearchGate*, vol. 137, no. 3, pp. 1–20, 2024.

[147] M. Malik and S. Bhushan, “Priority-Queue based Dynamic Scaling for Efficient Resource Allocation in Fog Computing,” *Netw. Internet Archit.*, vol. 12, no. 1, pp. 100–106, 2021.

[148] E. Wireless Commun. Netw., “Performance Prediction and Enhancement of 5G Network,” *SpringerOpen J. Wireless Commun. Netw.*, 2023.

[149] R. Yajima and A. Nakao, “End to End Latency Performance Evaluation for Building Low Latency and Low Jitter Applications in APN,” *IEICE Technical Report*, vol. 124, no. 179, NS2024-81, pp. 36-41, Sept. 2024.

[150] S. Zahid, M. Aazam, “Fog computing architecture, evaluation, and future research directions,” *IEEE Commun. Mag.*, vol. 56, no. 5, pp. 46–52, 2021.

[151] S. Ghosh, D. Gaur, P. Patel, J. Lin, G. Yu, X. Yang, and R. Ranjan, R. K. Naha, “Fog computing: Survey of trends, architectures, requirements, and research directions,” *IEEE Access*, vol. 6, pp. 47980–48009, 2020.

[152] K. Wang, G. Zhang, X. Chen, X. Liu, and M.-T. Zhou, Y. Yang, “Maximal energy efficient task scheduling in homogeneous fog networks,” *IEEE Internet Things J.*, vol. 5, no. 5, pp. 4076–4087, Oct. 2022.

[153] R. Roman, M. Gallucci, J. T. Soares, D. Santoro, and M. Villari, M. Fazio, “A Note on the Convergence of IoT, Edge, and Cloud Computing in Smart Cities,”

IEEE Cloud Comput., vol. 5, no. 5, pp. 22–24, 2021, doi: 10.1109/MCC.2021.053711663.

[154] R. Roman, M. Gallucci, J. T. Soares, D. Santoro, and M. Villari, M. Fazio, "A Note on the Convergence of IoT, Edge, and Cloud Computing in Smart Cities," *IEEE Cloud Comput.*, vol. 5, no. 5, pp. 22–24, 2020, doi: 10.1109/MCC.2020.053711663.

[155] J. Liang and H. Keung Kwan, "IIR Filter Design Using Constrained Multiobjective Cuckoo Search Algorithm," in *Proc. IEEE Canadian Conf. Electr. Comput. Eng. (CCECE)*, Quebec, QC, Canada, 2022, pp. 1–4, doi: 10.1109/CCECE.2022.8447811.

[156] M. Vu, N. Nguyen, Q. D. La, "Enabling intelligence in fog computing to achieve energy and latency reduction," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 3–9, 2023.

APPENDICES

Appendix 1: Questionnaire for Validating Variable Ranking by Industry Experts

Kindly Tick where best applies to your case (don't indicate your personal details)

PLEASE INDICATE YOUR GENDER: [Male] _____ [Female]

_____

1. Please TICK [√] your age bracket

[a] between 15 and 20 years [ ]

[b] Between 21 and 23 years [ ]

[c] Between 24 and 30 years [ ]

[d] From 31 years and above [ ]

2. What is your highest academic level of education?

[A] Secondary level [ ]

[b] Certificate level [ ]

[c] Diploma level [ ]

[d] Under graduate [ ]

[e] Post graduate [ ]

3. Are you familiar with Fog Computing?

Yes [ ] No [ ]

If Yes, have you ever interacted with fog device?

Yes [ ] No [ ]

4. How do you rate role of fog computing in the IOT ecosystem (edge-fog-cloud continuum)

[A] Not significant [ ]

[b] Significant [ ]

[c] Fairly significant [ ]

[d] Very significant [ ]

[e] Extremely significant [ ]

5. Please TICK [√] the course(s) that you have successfully completed in the Internet of Things (IOT) track.

[1] Huawei Certified Associate (IOT)	[ ]
[2] Huawei Certified Professional (IOT)	[ ]
[3] Huawei Certified Expert (IOT)	[ ]

[4] Cisco Certified Beginner (IOT)	[ ]
[5] Cisco Certified Intermediate (IOT)	[ ]
[6] Cisco Certified Advanced(IOT)	[ ]
[7] IBM Certified Basic (IOT)	[ ]
[8] IBM Certified Advanced (IOT)	[ ]

Do you have any other training in IoT? (Indicate)

…….

6. Indicate the ranking of the parameters listed below (1-6) by the level of importance in determining efficiency of fog computing in Internet of Things (IOT).

Variable	**Rank**				
	1	2	3	4	5
[a] Throughput					
[b] Latency					
[c] Channel utilization					
[d] Packet loss					
[e] Queue time					

7. List any other variable which in your view may be used to improve fog performance.

……

……

END

Appendix 2: Python Code to Generate Synthetic Data to Rank Variables

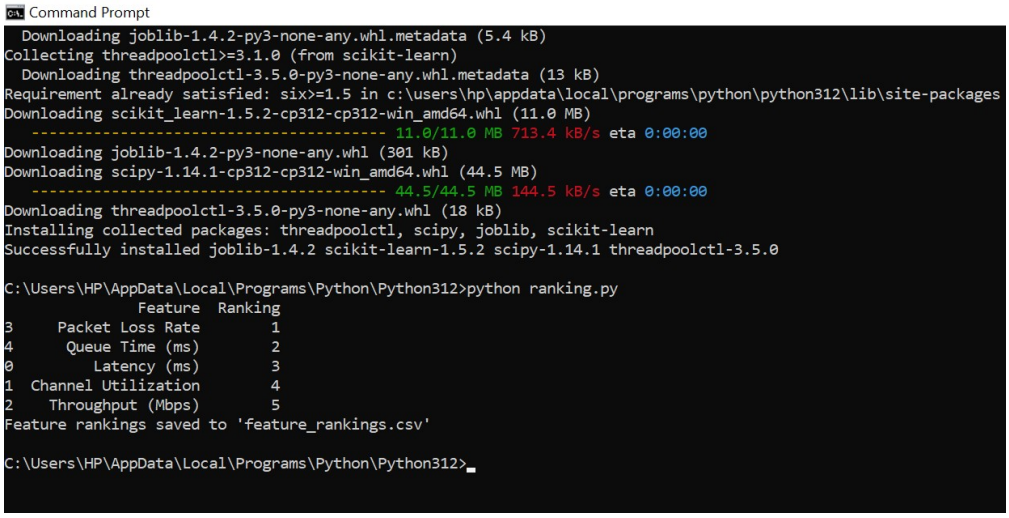
```
Import simpy
Import random
Import pandas as Pd
# Parameters
NUM_EVENTS = 1000
LATENCY_MEAN = 20 # Mean latency in MS
LATENCY_STD = 5    # Latency standard deviation
CHANNEL_UTILIZATION_MEAN = 0.5 # Mean utilization
CHANNEL_UTILIZATION_STD = 0.1    # Utilization standard deviation
THROUGHPUT_MEAN = 100 # Mean throughput in Mbps
THROUGHPUT_STD = 20     # Throughput standard deviation
RESPONSE_TIME_MEAN = 100 # Mean response time in MS
RESPONSE_TIME_STD = 10    # Response time standard deviation
PACKET_LOSS_RATE_MEAN = 0.01 # Mean packet loss rate
PACKET_LOSS_RATE_STD = 0.005   # Packet loss rate standard deviation
QUEUE_TIME_MEAN = 10   # Mean queue time in MS
QUEUE_TIME_STD = 2     # Queue time standard deviation
# Function to generate synthetic data
def generate_data(env):
   for _ in range(NUM_EVENTS):
      latency = random.gauss(LATENCY_MEAN, LATENCY_STD)
      channel_utilization = random.gauss(CHANNEL_UTILIZATION_MEAN,
CHANNEL_UTILIZATION_STD)
      throughput = random.gauss(THROUGHPUT_MEAN, THROUGHPUT_STD)
      response_time = random.gauss(RESPONSE_TIME_MEAN,
RESPONSE_TIME_STD)
      packet_loss_rate = random.gauss(PACKET_LOSS_RATE_MEAN,
PACKET_LOSS_RATE_STD)
      queue_time = random.gauss(QUEUE_TIME_MEAN, QUEUE_TIME_STD)
      yield env.timeout(1)  # Simulate time passing for each event
       # Save the generated data
      data.append({
         'Latency (MS)': max(0, latency),  # Ensure no negative latency
```

```
            'Channel Utilization': max(0, min(1, channel_utilization)),  # Utilization
between 0 and 1
            'Throughput (Mbps)': max(0, throughput),  # Ensure no negative throughput
            'Response Time (MS)': max(0, response_time),  # Ensure no negative
response time
            'Packet Loss Rate': max(0, min(1, packet_loss_rate)),  # Rate between 0 and 1
            'Queue Time (MS)': max(0, queue_time)  # Ensure no negative queue time
        })
# Simulation environment
env = simpy.Environment()
data = []
# Start the data generation process
env.process(generate_data(env))
env.run()
# Create a DataFrame and save to CSV
df = pd.DataFrame(data)
df.to_csv('fog_layer_performance_data.csv', index=False)
print("Synthetic data generated and saved to 'fog_layer_performance_data.csv'")
```

Appendix 3: Python Code to Rank Identified Variables

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split
# Load the dataset
data = pd.read_csv('fog_layer_performance_data.csv')
# Define features and target
# Assuming we want to predict 'Response Time' based on other variables
X = data.drop(columns=['Response Time (MS)'])
y = data['Response Time (MS)']
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Create a Random Forest model
model = RandomForestRegressor(n_estimators=100, random_state=42)
# Perform RFE
rfe = RFE(estimator=model, n_features_to_select=1)
rfe.fit(X_train, y_train)
# Get feature rankings
ranking = rfe.ranking_
feature_names = X.columns
# Create a DataFrame to display features and their rankings
ranking_df = pd.DataFrame({'Feature': feature_names, 'Ranking': ranking})
ranking_df = ranking_df.sort_values(by='Ranking')
# Output the rankings
print(ranking_df)
# Save the rankings to a CSV file
ranking_df.to_csv('feature_rankings.csv', index=False)
print("Feature rankings saved to 'feature_rankings.csv'")
```

Appendix 4: Screen Shot of variable Ranking Output



Appendix 5: Python Code to Test DBSCAN Performance

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.datasets import make_blobs
from sklearn.decomposition import PCA
# Step 1: Generate synthetic data for the variables
np.random.seed(42)
# Generate synthetic data: Packet Loss Rate, Queue Time, Latency, Channel Utilization, and Throughput
n_samples = 1000
X, _ = make_blobs(n_samples=n_samples, centers=3, n_features=5, random_state=42)
# Adding noise to the data to make it more realistic
X += np.random.normal(scale=0.5, size=X.shape)
# Step 2: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 3: Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=10)
dbscan_labels = dbscan.fit_predict(X_scaled)
# Step 4: Evaluate the clustering results
# Number of clusters and noise points
n_clusters = len(set(dbscan_labels)) - (1 if -1 in dbscan_labels else 0)
n_noise = list(dbscan_labels).count(-1)
print(f"Number of clusters found: {n_clusters}")
print(f"Number of noise points: {n_noise}")
# Silhouette score: Measures the quality of the clustering
sil_score = silhouette_score(X_scaled, dbscan_labels)
print(f"Silhouette Score: {sil_score:.4f}")
# If possible, use PCA to reduce dimensions for visual evaluation
```

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
# Step 5: Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_labels, cmap='viridis', s=10)
plt.colorbar(label='Cluster ID')
plt.title("DBSCAN Clustering on Synthetic Data (2D PCA)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
# Optional: Adjusted Rand Index (comparing with ground truth if known, or set manually)
# Here we are using random ground truth labels for demonstration purposes.
# You would compare with actual labels if available.
ground_truth = np.random.randint(0, 3, size=n_samples)  # Just for demonstration
ari_score = adjusted_rand_score(ground_truth, dbscan_labels)
print(f"Adjusted Rand Index: {ari_score:.4f}")
```

Appendix 8: Code to evaluate efficiencyof KNN

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
# Generate synthetic data
np.random.seed(42)
n_samples = 1000
data = {
    "Packet Loss Rate": np.random.uniform(0, 10, n_samples),  # % packet loss
    "Queue Time": np.random.uniform(0, 100, n_samples),  # MS
    "Latency": np.random.uniform(0, 200, n_samples),  # MS
    "Channel Utilization": np.random.uniform(0, 1, n_samples),  # 0 to 1
    "Throughput": np.random.uniform(10, 1000, n_samples),  # Mbps
```

```
}
# Generate labels (1: good performance, 0: poor performance)
# Assume some threshold conditions for classification
labels = (data["Packet Loss Rate"] < 2) & (data["Latency"] < 100) & (data["Throughput"] > 500)
data["Performance"] = labels.astype(int)
# Convert to a DataFrame
df = pd.DataFrame(data)
# Split into features and target
X = df[["Packet Loss Rate", "Queue Time", "Latency", "Channel Utilization", "Throughput"]]
y = df["Performance"]
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Initialize and train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
# Predict and evaluate
y_pred = knn.predict(X_test)
# Evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Poor", "Good"], yticklabels=["Poor", "Good"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Appendix 6: Python Code to Evaluate Efficiency of DBSCAN+KNN

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import DBSCAN
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
# Step 1: Generate synthetic data
np.random.seed(42)
n_samples = 1000
data = {
    "Packet Loss Rate": np.random.uniform(0, 10, n_samples),
    "Queue Time": np.random.uniform(0, 100, n_samples),
    "Latency": np.random.uniform(0, 200, n_samples),
    "Channel Utilization": np.random.uniform(0, 1, n_samples),
    "Throughput": np.random.uniform(10, 1000, n_samples),
}
# Assign labels: 1 (Good Performance), 0 (Poor Performance)
labels = (data["Packet Loss Rate"] < 2) & (data["Latency"] < 100) &
(data["Throughput"] > 500)
data["Performance"] = labels.astype(int)
# Convert to DataFrame
df = pd.DataFrame(data)
# Step 2: Scale features for DBSCAN
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.drop("Performance", axis=1))
# Step 3: Apply DBSCAN for clustering
dbscan = DBSCAN(eps=1.5, min_samples=10)
clusters = dbscan.fit_predict(X_scaled)
# Add clusters to DataFrame
df["Cluster"] = clusters
```

```
# Filter out noise (-1 cluster)
df_filtered = df[df["Cluster"] != -1]

# Step 4: Prepare data for KNN
X_filtered = df_filtered[["Packet Loss Rate", "Queue Time", "Latency", "Channel Utilization", "Throughput"]]
y_filtered = df_filtered["Performance"]
# Split filtered data into train-test sets
X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered, test_size=0.3, random_state=42)
# Train KNN on filtered data
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
# Predict and evaluate
y_pred = knn.predict(X_test)
# Efficiencymetrics
print("Hybrid Model (DBSCAN + KNN) Performance")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Poor", "Good"], yticklabels=["Poor", "Good"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
# Step 5: Baseline KNN Model for Comparison
X_baseline = df[["Packet Loss Rate", "Queue Time", "Latency", "Channel Utilization", "Throughput"]]
y_baseline = df["Performance"]
X_train_base, X_test_base, y_train_base, y_test_base = train_test_split(X_baseline, y_baseline, test_size=0.3, random_state=42)
```

```
knn_baseline = KNeighborsClassifier(n_neighbors=5)
knn_baseline.fit(X_train_base, y_train_base)
y_pred_base = knn_baseline.predict(X_test_base)
print("\nBaseline KNN Performance")
print("Accuracy:", accuracy_score(y_test_base, y_pred_base))
print("Classification Report:\n", classification_report(y_test_base, y_pred_base)).
```

Appendix 7: Python Code to Evaluate Efficiency of KNN+DBSCAN

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
# Step 1: Generate Synthetic Data
np.random.seed(42)
X, y = make_classification(
   n_samples=1000, n_features=5, n_informative=4, n_redundant=1, n_classes=2,
weights=[0.9, 0.1]
)
columns = ['Packet Loss Rate', 'Queue Time', 'Latency', 'Channel Utilization',
'Throughput']
data = pd.DataFrame(X, columns=columns)
data['Target'] = y
# Step 2: Split Data
X_train, X_test, y_train, y_test = train_test_split(
   data[columns], data['Target'], test_size=0.3, random_state=42, stratify=y)
# Step 3: Standardize Data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Step 4: Apply DBSCAN to Training Data
dbscan = DBSCAN(eps=0.8, min_samples=5)
labels = dbscan.fit_predict(X_train_scaled)
# Remove noise points labeled as -1
filtered_X_train = X_train_scaled[labels != -1]
filtered_y_train = y_train.values[labels != -1]
# Step 5: Train KNN on Filtered Data
```

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(filtered_X_train, filtered_y_train)
# Step 6: Evaluate Hybrid Model
y_pred = knn.predict(X_test_scaled)
# EfficiencyMetrics
print("Hybrid Model Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
# Evidence of Improvement (Baseline for comparison)
knn_baseline = KNeighborsClassifier(n_neighbors=5)
knn_baseline.fit(X_train_scaled, y_train)
baseline_pred = knn_baseline.predict(X_test_scaled)
print("\nBaseline KNN Performance:")
print(f"Accuracy: {accuracy_score(y_test, baseline_pred)}")
print("Classification Report:")
print(classification_report(y_test, baseline_pred))
```

Appendix 8: Python Code to Rate Individual Variables

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import DBSCAN
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Generate synthetic data
np.random.seed(42)
n_samples = 1000
data = {
    "Packet Loss Rate": np.random.uniform(0, 10, n_samples),
    "Queue Time": np.random.uniform(0, 100, n_samples),
    "Latency": np.random.uniform(0, 200, n_samples),
    "Channel Utilization": np.random.uniform(0, 1, n_samples),
    "Throughput": np.random.uniform(10, 1000, n_samples),
}
# Assign labels: 1 (Good Performance), 0 (Poor Performance)
labels = (data["Packet Loss Rate"] < 2) & (data["Latency"] < 100) &
(data["Throughput"] > 500)
data["Performance"] = labels.astype(int)
# Convert to DataFrame
df = pd.DataFrame(data)
# Step 2: Scale features for DBSCAN
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.drop("Performance", axis=1))
# Step 3: Apply DBSCAN for clustering
dbscan = DBSCAN(eps=1.5, min_samples=10)
clusters = dbscan.fit_predict(X_scaled)
# Add clusters to DataFrame
```

```
df["Cluster"] = clusters
# Filter out noise (-1 cluster)
df_filtered = df[df["Cluster"] != -1]
# Step 4: Prepare data for KNN
X_filtered = df_filtered[["Packet Loss Rate", "Queue Time", "Latency", "Channel Utilization", "Throughput"]]
y_filtered = df_filtered["Performance"]
# Split filtered data into train-test sets
X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered, test_size=0.3, random_state=42)
# Train KNN on filtered data
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
# Predict and evaluate
y_pred = knn.predict(X_test)
# Efficiencymetrics
print("Hybrid Model (DBSCAN + KNN) Performance")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Poor", "Good"], yticklabels=["Poor", "Good"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
# Step 5: Baseline KNN Model for Comparison
X_baseline = df[["Packet Loss Rate"]]
y_baseline = df["Performance"]
X_train_base, X_test_base, y_train_base, y_test_base = train_test_split(X_baseline, y_baseline, test_size=0.3, random_state=42)
knn_baseline = KNeighborsClassifier(n_neighbors=5)
```

```
knn_baseline.fit(X_train_base, y_train_base)
y_pred_base = knn_baseline.predict(X_test_base)
print("\nBaseline KNN Performance")
print("Accuracy:", accuracy_score(y_test_base, y_pred_base))
print("Classification Report:\n", classification_report(y_test_base, y_pred_base))
```

Appendix 9: Python Code to Simulate Efficiency of DBSCAN

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
# Step 1: Generate synthetic dataset
np.random.seed(42)
n_samples = 1000
n_features = 5  # Packet Loss Rate, Queue Time, Latency, Channel Utilization, Throughput
# Create synthetic data
X, _ = make_blobs(n_samples=n_samples, centers=5, n_features=n_features, random_state=42)
# Feature names
columns = ['Packet_Loss_Rate', 'Queue_Time', 'Latency', 'Channel_Utilization', 'Throughput']
# Convert to DataFrame
df = pd.DataFrame(X, columns=columns)
# Step 2: Preprocess data (scaling for DBSCAN)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 3: Apply DBSCAN
dbscan = DBSCAN(eps=1.5, min_samples=10)  # Adjust eps and min_samples based on dataset
clusters = dbscan.fit_predict(X_scaled)
# Add cluster labels to the DataFrame
df['Cluster'] = clusters
# Step 4: Evaluate DBSCAN performance
# Number of clusters (excluding noise points)
n_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)
n_noise = list(clusters).count(-1)
```

```
# Silhouette Score (only valid if at least 2 clusters are found)
if n_clusters > 1:
    silhouette_avg = silhouette_score(X_scaled, clusters)
    print(f"Silhouette Score: {silhouette_avg:.2f}")
else:
    print("Silhouette Score cannot be computed (fewer than 2 clusters).")
print(f"Number of clusters found: {n_clusters}")
print(f"Number of noise points: {n_noise}")
# Step 5: Visualize clusters
# Visualize in 2D by selecting two variables (e.g., Packet Loss Rate vs. Queue Time)
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap='viridis', s=50)
plt.title("DBSCAN Clustering Results")
plt.xlabel("Packet Loss Rate (scaled)")
plt.ylabel("Queue Time (scaled)")
plt.colorbar(scatter, label="Cluster")
plt.show()
# Step 6: Analyze fog efficiencymetrics for each cluster
metrics = df.groupby('Cluster')[['Packet_Loss_Rate', 'Queue_Time', 'Latency',
'Channel_Utilization', 'Throughput']].mean()
print("\nFog EfficiencyMetrics by Cluster:")
print(metrics)
```

Appendix 10: Python Code to Simulate Efficiency of KNN in YAFS

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
# Step 1: Generate synthetic dataset
np.random.seed(42)
n_samples = 1000
n_features = 5  # Packet Loss Rate, Queue Time, Latency, Channel Utilization, Throughput
n_classes = 4
# Generate synthetic data using make_blobs
X, y = make_blobs(n_samples=n_samples, centers=n_classes, n_features=n_features, random_state=42)
# Create a DataFrame
columns = ['Packet_Loss_Rate', 'Queue_Time', 'Latency', 'Channel_Utilization', 'Throughput']
df = pd.DataFrame(X, columns=columns)
df['Label'] = y  # Add ground truth labels
# Step 2: Preprocess data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 3: Train/test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
# Step 4: Train the KNN model
k_neighbors = 5
knn = KNeighborsClassifier(n_neighbors=k_neighbors)
knn.fit(X_train, y_train)
```

```
# Step 5: Make predictions
y_pred = knn.predict(X_test)
# Step 6: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"KNN Accuracy: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=range(n_classes), yticklabels=range(n_classes))
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
# Step 7: Analyze feature importance using distance metrics
# Evaluate the efficiencymetrics for the features in the test set
fog_metrics = pd.DataFrame(X_test, columns=columns).mean(axis=0)
print("\nFog EfficiencyMetrics:")
print(fog_metrics)
```

Appendix 11: Python Code to Simulate Efficiency of the DBSCAN+KNN Hybrid Model

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
# Step 1: Generate synthetic dataset
np.random.seed(42)
n_samples = 1000
n_features = 5  # Packet Loss Rate, Queue Time, Latency, Channel Utilization, Throughput
# Use make_blobs to create synthetic data clusters
X, y = make_blobs(n_samples=n_samples, centers=4, n_features=n_features, random_state=42)
# Scale features for DBSCAN
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 2: Apply DBSCAN for clustering
dbscan = DBSCAN(eps=1.5, min_samples=10)
clusters = dbscan.fit_predict(X_scaled)
# Add DBSCAN clusters to the synthetic dataset
synthetic_data = pd.DataFrame(X, columns=['Packet_Loss_Rate', 'Queue_Time', 'Latency', 'Channel_Utilization', 'Throughput'])
synthetic_data['Cluster'] = clusters
synthetic_data['Label'] = y  # Assign ground truth labels from make_blobs
# Filter out noise points (-1 cluster) for KNN training
filtered_data = synthetic_data[synthetic_data['Cluster'] != -1]
X_filtered = filtered_data[['Packet_Loss_Rate', 'Queue_Time', 'Latency', 'Channel_Utilization', 'Throughput']]
```

```
y_filtered = filtered_data['Label']
# Step 3: Train/Test split for KNN
X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered, test_size=0.3,
random_state=42)
# Train KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
# Predict and evaluate performance
y_pred = knn.predict(X_test)
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of KNN after DBSCAN filtering: {accuracy:.2f}")
# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
# Step 4: Efficiencymetrics for fog variables
fog_metrics = {
    'Packet Loss Rate': X_test['Packet_Loss_Rate'].mean(),
    'Queue Time': X_test['Queue_Time'].mean(),
    'Latency': X_test['Latency'].mean(),
    'Channel Utilization': X_test['Channel_Utilization'].mean(),
    'Throughput': X_test['Throughput'].mean()
}
print("\nFog EfficiencyMetrics:")
for metric, value in fog_metrics.items():
    print(f"{metric}: {value:.2f}")
# Step 5: Visualization of DBSCAN clusters
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap='viridis', s=50)
plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1 (Packet Loss Rate)')
plt.ylabel('Feature 2 (Queue Time)')
plt.colorbar(label='Cluster')
plt.show()
```

Appendix 12: Python code for evaluating Packet loss rate on the Hybrid model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Generate synthetic fog layer efficiencydata
np.random.seed(42)
n_samples = 500

# Feature: Packet Loss Rate
X = np.random.rand(n_samples, 1) * 5  # Simulating packet loss rate in a range of 0-5%

# Labels: 0 for normal traffic, 1 for anomalous traffic
# Introduce anomalies by setting high values
y = np.random.choice([0, 1], size=n_samples, p=[0.85, 0.15])
X[y == 1] += np.random.rand(sum(y == 1), 1) * 10  # Increase packet loss for anomalies

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply DBSCAN for preprocessing to detect and remove noise
dbscan = DBSCAN(eps=1.5, min_samples=10)
dbscan_labels = dbscan.fit_predict(X_scaled)
```

```
# Filter out noise points (DBSCAN labels -1 are considered noise)
mask = dbscan_labels != -1
X_filtered, y_filtered = X_scaled[mask], y[mask]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered, test_size=0.2,
random_state=42)

# Apply KNN on preprocessed data
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Hybrid Model Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)

# Data Visualization
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_scaled[:, 0], y=[0] * len(X_scaled), hue=dbscan_labels,
palette='viridis')
plt.title("DBSCAN Clustering Results in Fog Layer (Packet Loss Rate)")
plt.xlabel("Packet Loss Rate")
plt.yticks([])
plt.show()

plt.figure(figsize=(6, 5))
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal', 'Anomalous'], yticklabels=['Normal', 'Anomalous'])
plt.title("Confusion Matrix of KNN Classification in Fog Layer")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Scatter plot of packet loss rate after DBSCAN preprocessing
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_filtered[:, 0], y=[0] * len(X_filtered), hue=y_filtered, palette='coolwarm', alpha=0.7)
plt.title("Scatter Plot of Packet Loss Rate After DBSCAN Preprocessing")
plt.xlabel("Packet Loss Rate")
plt.yticks([])
plt.show()
```

Appendix 13: Python Code for Evaluating Queue Time on the Hybrid Model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Generate synthetic fog layer efficiencydata
np.random.seed(42)
n_samples = 500

# Feature: Queue Time
X = np.random.rand(n_samples, 1) * 100  # Simulating queue time in a range of 0-100 ms

# Labels: 0 for normal traffic, 1 for anomalous traffic
# Introduce anomalies by setting high values
y = np.random.choice([0, 1], size=n_samples, p=[0.85, 0.15])
X[y == 1] += np.random.rand(sum(y == 1), 1) * 200  # Increase queue time for anomalies

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply DBSCAN for preprocessing to detect and remove noise
dbscan = DBSCAN(eps=1.5, min_samples=10)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Filter out noise points (DBSCAN labels -1 are considered noise)
```

```
mask = dbscan_labels != -1
X_filtered, y_filtered = X_scaled[mask], y[mask]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered, test_size=0.2,
random_state=42)

# Apply KNN on preprocessed data
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Hybrid Model Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)

# Data Visualization
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_scaled[:, 0], y=[0] * len(X_scaled), hue=dbscan_labels,
palette='viridis')
plt.title("DBSCAN Clustering Results in Fog Layer (Queue Time)")
plt.xlabel("Queue Time")
plt.yticks([])
plt.show()

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal',
'Anomalous'], yticklabels=['Normal', 'Anomalous'])
plt.title("Confusion Matrix of KNN Classification in Fog Layer")
```

```
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Scatter plot of queue time after DBSCAN preprocessing
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_filtered[:, 0], y=[0] * len(X_filtered), hue=y_filtered,
palette='coolwarm', alpha=0.7)
plt.title("Scatter Plot of Queue Time After DBSCAN Preprocessing")
plt.xlabel("Queue Time")
plt.yticks([])
plt.show()
```

Appendix 14: Python Code for Evaluating Latency on the Hybrid Model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Generate synthetic fog layer efficiencydata
np.random.seed(42)
n_samples = 500

# Feature: Latency
X = np.random.rand(n_samples, 1) * 200  # Simulating latency in a range of 0-200 ms

# Labels: 0 for normal traffic, 1 for anomalous traffic
# Introduce anomalies by setting high values
y = np.random.choice([0, 1], size=n_samples, p=[0.85, 0.15])
X[y == 1] += np.random.rand(sum(y == 1), 1) * 400  # Increase latency for anomalies

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply DBSCAN for preprocessing to detect and remove noise
dbscan = DBSCAN(eps=1.5, min_samples=10)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Filter out noise points (DBSCAN labels -1 are considered noise)
mask = dbscan_labels != -1
X_filtered, y_filtered = X_scaled[mask], y[mask]
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered, test_size=0.2,
random_state=42)

# Apply KNN on preprocessed data
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Hybrid Model Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)

# Data Visualization
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_scaled[:, 0], y=[0] * len(X_scaled), hue=dbscan_labels,
palette='viridis')
plt.title("DBSCAN Clustering Results in Fog Layer (Latency)")
plt.xlabel("Latency")
plt.yticks([])
plt.show()

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal',
'Anomalous'], yticklabels=['Normal', 'Anomalous'])
plt.title("Confusion Matrix of KNN Classification in Fog Layer")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
plt.show()

# Scatter plot of latency after DBSCAN preprocessing
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_filtered[:, 0], y=[0] * len(X_filtered), hue=y_filtered,
palette='coolwarm', alpha=0.7)
plt.title("Scatter Plot of Latency After DBSCAN Preprocessing")
plt.xlabel("Latency")
plt.yticks([])
plt.show()
```

Appendix 15: Python Code for Evaluating Channel Utilization on the Hybrid Model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Generate synthetic fog layer efficiencydata
np.random.seed(42)
n_samples = 500

# Feature: Channel Utilization
X = np.random.rand(n_samples, 1) * 1  # Simulating channel utilization in a range of 0-1 (0% to 100%)

# Labels: 0 for normal traffic, 1 for anomalous traffic
# Introduce anomalies by setting high values
y = np.random.choice([0, 1], size=n_samples, p=[0.85, 0.15])
X[y == 1] += np.random.rand(sum(y == 1), 1) * 2  # Increase channel utilization for anomalies

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply DBSCAN for preprocessing to detect and remove noise
dbscan = DBSCAN(eps=1.5, min_samples=10)
dbscan_labels = dbscan.fit_predict(X_scaled)
```

```
# Filter out noise points (DBSCAN labels -1 are considered noise)
mask = dbscan_labels != -1
X_filtered, y_filtered = X_scaled[mask], y[mask]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered, test_size=0.2,
random_state=42)

# Apply KNN on preprocessed data
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Hybrid Model Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)

# Data Visualization
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_scaled[:, 0], y=[0] * len(X_scaled), hue=dbscan_labels,
palette='viridis')
plt.title("DBSCAN Clustering Results in Fog Layer (Channel Utilization)")
plt.xlabel("Channel Utilization")
plt.yticks([])
plt.show()

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal',
'Anomalous'], yticklabels=['Normal', 'Anomalous'])
```

```
plt.title("Confusion Matrix of KNN Classification in Fog Layer")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Scatter plot of channel utilization after DBSCAN preprocessing
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_filtered[:, 0], y=[0] * len(X_filtered), hue=y_filtered,
palette='coolwarm', alpha=0.7)
plt.title("Scatter Plot of Channel Utilization After DBSCAN Preprocessing")
plt.xlabel("Channel Utilization")
plt.yticks([])
plt.show()
```

Appendix 16: Python Code for Evaluating Throughput on the Hybrid Model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Generate synthetic fog layer efficiencydata
np.random.seed(42)
n_samples = 500

# Feature: Throughput
X = np.random.rand(n_samples, 1) * 100  # Simulating throughput in a range of 0-100 Mbps

# Labels: 0 for normal traffic, 1 for anomalous traffic
# Introduce anomalies by setting low values
y = np.random.choice([0, 1], size=n_samples, p=[0.85, 0.15])
X[y == 1] -= np.random.rand(sum(y == 1), 1) * 50  # Decrease throughput for anomalies

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply DBSCAN for preprocessing to detect and remove noise
dbscan = DBSCAN(eps=1.5, min_samples=10)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Filter out noise points (DBSCAN labels -1 are considered noise)
```

```
mask = dbscan_labels != -1
X_filtered, y_filtered = X_scaled[mask], y[mask]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered, test_size=0.2,
random_state=42)

# Apply KNN on preprocessed data
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Hybrid Model Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)

# Data Visualization
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_scaled[:, 0], y=[0] * len(X_scaled), hue=dbscan_labels,
palette='viridis')
plt.title("DBSCAN Clustering Results in Fog Layer (Throughput)")
plt.xlabel("Throughput")
plt.yticks([])
plt.show()

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal',
'Anomalous'], yticklabels=['Normal', 'Anomalous'])
plt.title("Confusion Matrix of KNN Classification in Fog Layer")
```

```
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Scatter plot of throughput after DBSCAN preprocessing
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_filtered[:, 0], y=[0] * len(X_filtered), hue=y_filtered,
palette='coolwarm', alpha=0.7)
plt.title("Scatter Plot of Throughput After DBSCAN Preprocessing")
plt.xlabel("Throughput")
plt.yticks([])
plt.show()
```

Appendix 17: Python Code for Evaluating Ensemble KNN on YAFS

```
from yafs.application import Application
from yafs.topology import Topology
from yafs.placement import Placement
from yafs.selection import Selection
from yafs.distribution import deterministic_distribution
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import BaggingRegressor
import numpy as np
import pandas as pd

# Step 1: Create synthetic dataset
# Efficiency variables: Packet Loss Rate, Queue Time, Latency, Channel Utilization, Throughput
np.random.seed(42)
data_size = 150
# Create random data representing fog system metrics
X = np.random.rand(data_size, 5)
# Synthetic performance target: combination of metrics with random noise
y = (
    # Packet loss rate (negative influence)
    # Queue time
    # Latency
    # Channel utilization
    # Throughput (negative influence)
    np.random.normal(0, 0.05, data_size)
)

# Step 2: Train Ensemble KNN (Bagging over KNN)
base_knn = KNeighborsRegressor(n_neighbors=5)
ensemble_knn = BaggingRegressor(base_estimator=base_knn, n_estimators=15, random_state=42)
ensemble_knn.fit(X, y)
# Generate test samples (simulated fog environment metrics)
test_data = np.random.rand(10, 5)
predicted_efficiency = ensemble_knn.predict(test_data)
# Step 3: Simulate a Minimal Fog Environment in YAFS
topo = Topology()
topo_json = {
    "entity": [
        {"id": 0, "model": "cloud"},
        {"id": 1, "model": "fog"},
        {"id": 2, "model": "edge"}
```

```
    ],
    "link": [
        {"s": 0, "d": 1},
        {"s": 1, "d": 2}
    ]
}
topo.load(topo_json)
# Define simple application structure
app = Application(name="EnsembleKNN_FogSimulation")
app.set_modules(modules=[
    {"Module": "Sensor", "Type": Application.TYPE_SOURCE},
    {"Module": "Processor", "Type": Application.TYPE_MODULE},
    {"Module": "Actuator", "Type": Application.TYPE_SINK}
])
# Step 4: Compute Average Fog Efficiency for Each Variable
efficiency_variables = ["Packet Loss Rate", "Queue Time", "Latency", "Channel
Utilization", "Throughput"]
# Compute synthetic average efficiency influence for demonstration
avg_contributions = {
    "Packet Loss Rate": round(np.mean(test_data[:, 0]), 2),
    "Queue Time": round(np.mean(test_data[:, 1]), 2),
    "Latency": round(np.mean(test_data[:, 2]), 2),
    "Channel Utilization": round(np.mean(test_data[:, 3]), 2),
    "Throughput": round(np.mean(test_data[:, 4]), 2)
}
# Create DataFrame of simulated results
df_results = pd.DataFrame(avg_contributions, index=["ENSEMBLE KNN"]).T
# Save to file (optional)
df_results.to_csv("ensemble_knn_fog_efficiency_results.csv")
# Step 5: Silent simulation complete
# The results are stored in 'ensemble_knn_fog_efficiency_results.csv'
# No direct print statements are included to maintain silent execution.
```

Appendix 18: MUT Research Authorization letter



MURANG'A UNIVERSITY OF TECHNOLOGY

Office of the Director, Postgraduate Studies

E-mail: bps@mut.ac.ke
Website: www.mut.ac.ke
P.O. Box 75-10200
Murang'a, Kenya

Ref: MUT/ARP/PGS/19/2020 /VOL.1 **Date:** 10th July, 2024

Mr. Watson Musyoki Kanuku -SC500/5564/2021
School of Computing and Information Technology,
Murang'a University of Technology,
P.O. Box 75-10,200
Murang'a.

Dear Mr. Kanuku,

RE: APPROVAL OF RESEARCH PROPOSAL AND SUPERVISORS.

I am pleased to inform you that the Directorate of Postgraduate Studies on 1st July 2024 considered and approved your Doctoral research proposal entitled ***"A hybrid model for Improving the performance of fog layer in the internet of things"*** and appointed the following as supervisors:

1. Dr. Stephen Njenga - Murang'a University of Technology
2. Dr. George Musumba – Dedan Kimathi University of Technology

You may now proceed with your data collection subject to obtaining research permit from NACOSTI, if required. You should also begin consulting your supervisors and submit through them quarterly progress reports to the Director Postgraduate Studies through your CoD and School Dean. Progress Reports forms can be accessed in the Student Portal.

It is the policy and regulations of the University that you observe deadlines. The guidelines on Postgraduate supervision can be accessed in the post graduate Handbook.

Your responsibilities as a student will include, among others;

i. Maintain regular consultation with your supervisor(s), at least once a month

ii. Submit quarterly reports on time, through your supervisors, CoD, Dean and to the Director of Postgraduate Studies;
iii. Ensure quality work all through;
iv. Present your research findings at 2-3 seminars/conferences prior to thesis examination.
v. Publish two articles from your research findings in a refereed journal prior to thesis examination.

For any further clarification please contact the undersigned.

Yours Sincerely,

Prof. Geoffrey Muchiri, PhD
Director, Postgraduate Studies
CM/PPH

DIRECTORATE OF POSTGRADUATE STUDIES
MURANG'A UNIVERSITY OF TECHNOLOGY
10 JUL 2024
P. O. Box 75 - 10200, MURANG'A
Tel: 0771463515, Email: info@mut.ac.ke

Copy to:
1. Registrar – Academics and Students Affairs
2. Dean, School of Computing and Information Technology

Appendix 19: NACOSTI Research license

REPUBLIC OF KENYA

NATIONAL COMMISSION FOR SCIENCE,TECHNOLOGY & INNOVATION

Ref No: **287446**

Date of Issue: **17/February/2024**

RESEARCH LICENSE



This is to Certify that Mr.. Watson Musyoki Kanuku of Murang'a University of Technology, has been licensed to conduct research as per the provision of the Science, Technology and Innovation Act, 2013 (Rev.2014) in Machakos, Mombasa, Muranga, Nairobi on the topic: A HYBRID MODEL FOR IMPROVING THE EFFICIENCY OF FOG LAYER IN THE INTERNET OF THINGS for the period ending : 17/February/2025.

License No: **NACOSTI/P/24/4184746**

287446

Applicant Identification Number

Ag. Director General
NATIONAL COMMISSION FOR SCIENCE,TECHNOLOGY & INNOVATION

Verification QR Code

NOTE: This is a computer generated License. To verify the authenticity of this document, Scan the QR Code using QR scanner application.

See overleaf for conditions

THE SCIENCE, TECHNOLOGY AND INNOVATION ACT, 2013 (Rev. 2014)
Legal Notice No. 108: The Science, Technology and Innovation (Research Licensing) Regulations, 2014

The National Commission for Science, Technology and Innovation, hereafter referred to as the Commission, was the established under the Science, Technology and Innovation Act 2013 (Revised 2014) herein after referred to as the Act. The objective of the Commission shall be to regulate and assure quality in the science, technology and innovation sector and advise the Government in matters related thereto.

CONDITIONS OF THE RESEARCH LICENSE

1. The License is granted subject to provisions of the Constitution of Kenya, the Science, Technology and Innovation Act, and other relevant laws, policies and regulations. Accordingly, the licensee shall adhere to such procedures, standards, code of ethics and guidelines as may be prescribed by regulations made under the Act, or prescribed by provisions of International treaties of which Kenya is a signatory to.
2. The research and its related activities as well as outcomes shall be beneficial to the country and shall not in any way; i. Endanger national security
   ii. Adversely affect the lives of Kenyans
   iii. Be in contravention of Kenya's international obligations including Biological Weapons Convention (BWC), Comprehensive Nuclear-Test-Ban Treaty Organization (CTBTO), Chemical, Biological, Radiological and Nuclear (CBRN).
   iv. Result in exploitation of intellectual property rights of communities in Kenya
   v. Adversely affect the environment
   vi. Adversely affect the rights of communities
   vii. Endanger public safety and national cohesion
   viii. Plagiarize someone else's work
3. The License is valid for the proposed research, location and specified period.
4. Neither the license nor any rights thereunder are transferable.
5. The Commission reserves the right to cancel the research at any time during the research period if in the opinion of the Commission the research is not implemented in conformity with the provisions of the Act or any other written law.
6. The Licensee shall inform the relevant County Director of Education, County Commissioner and County Governor before commencement of the research.
7. Excavation, filming, movement, and collection of specimens are subject to further necessary clearance from relevant Government Agencies.
8. The License does not give authority to transfer research materials.
9. The Commission may monitor and evaluate the licensed research project for the purpose of assessing and evaluating compliance with the conditions of the License.
10. The Licensee shall submit one hard copy, and upload a soft copy of their final report (thesis) onto a platform designated by the Commission within one year of completion of the research.
11. The Commission reserves the right to modify the conditions of the License including cancellation without prior notice.
12. Research, findings and information regarding research systems shall be stored or disseminated, utilized or applied in such a manner as may be prescribed by the Commission from time to time.
13. The Licensee shall disclose to the Commission, the relevant Institutional Scientific and Ethical Review Committee, and the relevant national agencies any inventions and discoveries that are of National strategic importance.
14. The Commission shall have powers to acquire from any person the right in, or to, any scientific innovation, invention or patent of strategic importance to the country.
15. Relevant Institutional Scientific and Ethical Review Committee shall monitor and evaluate the research periodically, and make a report of its findings to the Commission for necessary action.

National Commission for Science, Technology and
Innovation(NACOSTI),
Off Waiyaki Way, Upper Kabete,
P. O. Box 30623 - 00100 Nairobi, KENYA
Telephone: 020 4007000, 0713788787, 0735404245
E-mail: dg@nacosti.go.ke
Website: www.nacosti.go.ke

Appendix 20: Publications

This theses has produced the following publications:

Kanuku M, Njenga S, Musumba G. **"**A Systematic Review of Key Variables Affecting Fog Computing Performance in IoT Applications: International Journal of Science and Research (IJSR), https://www.ijsr.net/search_index_results_paperid.php?id=SR241017130635. Volume 13 Issue 10, October 2024.

Kanuku M, Njenga Stephen, Musumba George, "Ranking and Validation of Variables Determining Fog Computing Performance in IoT Applications", International Journal of Science and Research (IJSR), Volume 14 Issue 4, April 2025, pp. 1057-1063, https://www.ijsr.net/getabstract.php?paperid=SR25408144751, DOI: **https://www.doi.org/10.21275/SR25408144751.**