Smart IoT Based Remote Health Monitoring System



SUPERIOR UNIVERSITY

A Thesis submitted in the partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN
ELECTRICAL ENGINEERING

Submitted By
Wasif Arshad
SU92-MSEEW-F23-006

Session: 2023-2025

Supervised By
Dr. Mustafa Shakir

DEPARTMENT OF ELECTRICAL ENGINEERING
FACULTY OF ENGINEERING & TECHNOLOGY
SUPERIOR UNIVERSITY
LAHORE, PAKISTAN

AUTHOR'S DECLARATION

I, Mr. Wasif Arshad a student of "**Master of Science in Electrical Engineering**", at "**Faculty of Engineering & Technology**", the Superior University, Lahore, hereby declare that this thesis titled, "**Smart IoT Based Remote Health Monitoring System**" is my own research work and has not been submitted, published, or printed elsewhere in Pakistan or abroad. Additionally, I will not use this thesis to obtain any degree other than the one stated above.

I fully understand that if my statement is found to be incorrect at any stage, including after the award of the degree, the university has the right to revoke my MS degree.



Signature of Student: _____

Name of Student: Wasif Arshad

Roll Number: SU92-MSEEW-F23-006

Date: 09-12-25

PLAGIARISM UNDERTAKING

I solemnly declare that the research work presented in this thesis titled, "**Smart IoT-Based Remote Health Monitoring System**" is solely my research work, and that the entire thesis has been completed by me, with no significant contribution from any other person or institution. Any small contribution, wherever taken, has been duly acknowledged.

I understand the zero-tolerance policy of the HEC and Superior University, Lahore, towards plagiarism. Therefore, I, as the author of the above-titled thesis, declare that no portion of my thesis has been plagiarized and that all material used from other sources has been properly acknowledged, cited, and referenced.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis, even after the award of MS degree, the University reserves the right to revoke my degree, and that HEC and the University have the right to publish my name on the HEC/University website for submitting a plagiarized thesis.

Signature of Student: 

Name of Student: Wasif Arshad

Signature Supervisor: _____

Supervisor Name: Dr. Mustafa Shakir

CERTIFICATE OF RESEARCH COMPLETION

It is certified that this thesis titled, "**Smart IoT Based Remote Health Monitoring System**", submitted by Wasif Arshad, Roll No, SU92-MSEEW-F23-006, for MS "**Master of Science in Electrical Engineering**" at "**Faculty of Engineering & Technology**", Superior University, is an original research work and contains satisfactory material to be eligible for evaluation by the Examiner(s) for the award of the above-stated degree.

No part of this dissertation has been submitted anywhere else for any other degree.

Supervisor's Signature: _____

Supervisor's Name: **Dr. Mustafa Shakir**

Designation: Chairman / Associate Professor (Department of Electrical Engineering)

Co-Supervisor's Signature: _____

Co-Supervisor's Name: _____

Designation:

CERTIFICATE OF APPROVAL

It is certified that the research work contained in this thesis, titled "**Smart IoT Based Remote Health Monitoring System**" was conducted by "**Wasif Arshad**" under the supervision of "**Dr. Mustafa Shakir**" and is adequate for the award of "**Master of Science in Electrical Engineering**". No part of this thesis has been submitted anywhere else to any other degree. This thesis is submitted to the "**Faculty of Engineering & Technology**", The Superior University, Lahore in partial fulfillment of the requirements for the degree.

Student Name: Wasif Arshad Signature: 

Examination Committee:

a) Session Chair: Signature: _____

b) External Examiner

Name: Signature: _____
Designation:
University Name:

c) Internal Examiner

Name: Signature: _____
Designation:
University Name:

d) Supervisor Name: Dr. Mustafa Shakir Signature: _____

e) Head of Department: Dr. Manzoor Elahi Signature: _____

f) Name of the Dean: Dr. Muhammad Zafrullah Signature:_____

g) COE: Signature: _____

DEDICATION

To my beloved family, teachers and friends.

ACKNOWLEDGEMENTS

“I would like to express my sincere gratitude to my supervisor, **Dr. Mustafa Shakir**, for his invaluable guidance, encouragement, and continuous support throughout the course of this research. I am also deeply grateful to all the instructors in the Department of Electrical Engineering Especially **Dr. Manzoor Elahi**, **Dr. Rao Muhammad Asif** for their insightful teaching and academic guidance, which greatly contributed to my knowledge and skills. Lastly, I would like to extend my appreciation to my colleagues and friends, whose cooperation and assistance have been a constant source of motivation during this work.”

Thank you,
Wasif Arshad

PUBLICATIONS LIST

It is certified that the following publication(s) have been made of the research work that has been carried out for this thesis

Journal Article:

Wasif Arshad, Murtaza Anjum, and Mustafa Shakir, “Real-time blood glucose prediction using SARIMAX and edge computing,” *Academia International Journal for Social Sciences*, Academia Pvt. Ltd., vol. 4, no. 3, pp. 476–484, Jul. 2025. doi: 10.63056/ACAD.004.03.0476.

Conference Paper

Wasif Arshad, Mustafa Shakir, Murtaza Anjum, Muhammad Faiq, and Zafarullah Khan, “Evaluation of SARIMAX and LSTM for continuous glucose monitoring data prediction,” submitted to *Proc. 11th Int. Conf. Eng. Emerging Technol. (ICEET 2025)*, Kuala Lumpur, Malaysia, Oct. 22–23, 2025. Submitted Aug. 30, 2025.

TABLE OF CONTENTS

AUTHOR'S DECLARATION I
PLAGIARISM UNDERTAKING II
CERTIFICATE OF RESEARCH COMPLETION III
CERTIFICATE OF APPROVAL IV
DEDICATION V
ACKNOWLEDGEMENTS VI
PUBLICATIONS LIST VII
ABBREVIATIONS & ACRONYMS XI
LIST OF TABLES XII
LIST OF FIGURES XIII
ABSTRACT XIV

CHAPTER 1 1
INTRODUCTION 1
1.1 Background 1
1.2 Problem Statement 2
1.3 Research Motivation 4
1.4 Research Objectives 5
1.5 Main Contributions 6
1.6 Scope Of Study 7
1.7 Used Environment 8

CHAPTER 2 11
LITERATURE REVIEW 11
2.1 Introduction 11
2.2 Transforming HealthCare Through Remote Patient Monitoring 11
2.3 Hybrid Edge Cloud Architecture In Remote Patient Monitoring 13
2.4 Time Series Forecasting In Health Care 15
2.5 Alert Generation and Dashboard Integration 17
2.6 Data Security in Healthcare IoT 18
2.7 Current Challenges and Research Opportunities in RPM Architecture 20

CHAPTER 3 23
METHODOLOGICAL FRAMEWORK AND SYSTEM ARCHITECTURE 23
3.1 Introduction 23
3.2 Research Methodology 24
3.3 System Objectives and Optimization Functions 28
3.4 System Overview 30
3.5 System Architecture 32
3.6 Tools and Technologies 40
3.7 Forecasting Framework 51
3.8 AWS-Based Automation and Alert System 70

CHAPTER 4 74
SIMULATIONS AND RESULTS 74
4.1 Introduction 74
4.2 Simulation Environment 75
4.3 System Design and Implementation in Simulation 77
4.4 System Testing 81
4.5 Accuracy Comparison Of SARIMAX & LSTM On Simulated Data 85
4.6 Forecasting Models Execution On Microcontroller 104
4.7 Alert Generation and AWS Integration Results 107
4.8 Dashboard & Graphics 114
4.9 Simulation Of AES Encryption On Data Security 115
4.10 Performance Evaluation & Benchmarking 117
4.11 Comparative Analysis 120
4.12 Challenges faced During Simulation 120

CHAPTER 5 123
Conclusion & Future Work 123
5.1 Overview 123
5.2 Summary Of Work 123
5.3 Major Findings 124
5.4 Limitations 124
5.5 Conclusion 125
5.6 Future Work & Improvements 126

5.7 Main Contributions Of Study 126

REFERENCES 127

APPENDIX A 135

A-1 Pyhton Script Simulated Blood Glucose Data 135

A-2 Pyhton Script SARIMAX Forecasting Blood Glucose Data 138

A-3 Pyhton Script Historical & SARIMAX Forecasted Data Comparison 141

A-4 Pyhton Script LSTM Forecasting Blood Glucose Data 143

A-5 Pyhton Script Historical & LSTM Forecasted Data Comparison 146

A-6 Pyhton Script Performance & Accuracy Comparison SARIMAX & LSTM 148

A-7 Pyhton Script Data Uploader From Microcontroller To AWS S3 Bucket 150

A-8 Pyhton Script Configuring Microcontroller as MODBUS Server 151

ABBREVIATIONS & ACRONYMS

IoT	Internet Of Things
RPM	Remote patient monitoring
SARIMAX	Seasonal Autoregression Integrated Moving Average with eXogenous Variable
ARIMA	Autoregression Integrated Moving Average
LSTM	Long Short Term Memory
AWS	Amazon Web Service
UNSDG's	United Nations Sustainable Development Goals
SNS	Simple Notification Service
HL7	Health Level 7
FHIR	Fast Health Care Interoperability Resources
HIPPA	Health Insurance Portability & Accountability Act
GDPR	General Data Protection Regulation
HMI	Human Machine Interface
GPIO	General Purpose Input Output
I2C	Inter Integrated Circuit
IDE	Integrated Development Environment
MQTT	Message Queuing Telemetry Transport
HTTP	Hyper Text Transfer Protocol
IAM	Identity & Access Management
AES	Advance Encryption Standard
ASCII	American Standard Code for Information Interchange
EHR	Electronic Health Record
JWT	JSON Web Token
ECB	Electronic Code Book
GUI	Graphics User Interface
CLI	Command Line Interface
PCIe	Peripheral Component Interconnect Express

LIST OF TABLES

Table 2.1 Research Gaps and Challenges 22
Table 3.1 Key Features Comparison ARIMA, SARIMAX, and LSTM 67
Table 3.2 Deployemt-Oriented Comparison of SARIMAX & LSTM 68

LIST OF FIGURES

Figure 3.1: Patient Health Monitoring System Architecture with Edge Cloud Connectivity 32
Figure 3.2: Visual Code Studio Simulator for Python 45
Figure 3.3: Programming Environment in Visual Code Studio Showing Required Libraries 46
Figure 3.4: AWS Console Showing Essential Services Used In Real-World Health Monitoring 48
Figure 3.5: RC Studio Graphical User Interface Developer Software 50
Figure 3.6: SARIMAX Model Architecture 55
Figure 3.7: LSTM Model Flow Chart 63
Figure 3.8: Trigger Logic Inside AWS Lambda Function 71
Figure 3.9:.SNS Configuration for Email Alert 72
Figure 3.10: Uploaded Data Sets Inside S3 Bucket 73
Figure 4.1: Data Plots Showing Blood Glucose Data 87
Figure 4.2: Plot Showing SARIMAX Forecasting Results On One Month Training Data 89
Figure 4.3: Plot Showing SARIMAX Forecasting Results On Six Months Training Data 90
Figure 4.4: Plot Showing SARIMAX Forecasting Results On Twelve Months Training Data .. 91
Figure 4.5: Plot Showing Actual & Historical Data With % Error For Small Data Set 92
Figure 4.6: Plot Showing Actual & Historical Data With % Error For Medium Data Set 93
Figure 4.7: Plot Showing Actual & Historical Data With % Error For Large Data Set 93
Figure 4.8: Plot Showing LSTM Forecasting Result On One Month Training Data 95
Figure 4.9: Plot Showing LSTM Forecasting Result On Six Months Training Data 96
Figure 4.10: Plot Showing LSTM Forecasting Result On Twelve Months Training Data..... 97
Figure 4.11: Plot Showing Actual & Historical Data With % Error For Small Data Set 98
Figure 4.12: Plot Showing Actual & Historical Data With % Error For Medium Data Set 99
Figure 4.13 Plot Showing Actual & Historical Data With % Error For Medium Data Set 99
Figure 4.14: Comparison Of SARIMAX & LSTM Forecast Using One Month Training Data 101
Figure 4.15: Comparison Of SARIMAX & LSTM Forecast Using Six Month Training Data . 102
Figure 4.16: Comparison Of SARIMAX & LSTM Forecast Using Twelve Month Training Data 102
Figure 4.17: Raspberry PI 5 Command Line Interface Showing Python Script Implementation 104
Figure 4.18: Execution Of SARIMAX Python Script Showing Showing Results In Plot 104
Figure 4.19: Execution Of Python Script In Raspberry PI 5 Showing Result In Plot 105
Figure 4.20: Execution Of Python Script In CLI Of Controller 106
Figure 4.21: Execution Of Python Script In GUI Of Controller 107
Figure 4.22: AWS Console Showing Recently Used Services 108
Figure 4.23: Trigger Alert Script On AWS Lambda Function 109
Figure 4.24: SNS Configuration On AWS SNS Page 110
Figure 4.25: Received Emails Log On Configured ID 111
Figure 4.26: Exection Of Python Script To Upload Data To AWS 112
Figure 4.27: Uploaded Data Sets At AWS S3 Bucket 112
Figure 4.28: Graphics User Interface & Dashboard Design 114
Figure 4.29: Simulation Of AES Encryption Using Python in VS Code 116

ABSTRACT

This study outlines how a hybrid edge-cloud remote patient monitoring framework (RPM) can be designed and implemented that would combine the Internet of Things (IoT), machine learning, and cloud automation into a framework that would facilitate real-time health prediction and alert production. The system uses simulated sensors alongside a Raspberry 5 controller to measure some vital physiological parameters, i.e. heart rate and blood glucose. Data collected at edges are preprocessed and analyzed on the local level, and cloud integration with the AWS Lambda and Simple Notification Service (SNS) make it possible to perform automatic alerts and be scalable. The hybrid architecture will provide effective data processing, low latency, and stable remote healthcare management.

In order to increase predictive precision, two predictive time-series models were used, the SARIMAX and the Bidirectional Long Short-Term Memory (LSTM) to generate synthetic health data. SARIMAX model was more effective to absorb seasonal and trend-based fluctuations especially on short data sets and the LSTM model proved to be better to adapt to the non-linear and complex patterns of time. Comparative analysis confirmed that the LSTM model provided reduced forecasting errors, especially when using larger data sets, thus replying the strength of the model when used to predict the health of the individual in the long-term. Both models have been compared based on two key parameters the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) and the findings indicate that the accuracy of the prediction increased with the increase in the size of the dataset.

The interactive dashboard built through RC-Studio and RC-graphics was also presented in the study and finally deployed to a web controller that was integrated to local micro controller configured as a Modbus device over a Modbus TCP/IP physical network. This dashboard has been used to visualize real-time data, which has improved the interpretation of health measures and system behavior. The effective combination of Raspberry integrated with web controller shows that the system can be used as scalable real life example of IoT healthcare prototype.

On the whole, the suggested system justifies the fact that the combination of edge computing, machine learning, and cloud services can build a powerful, smart RPM system. The study designs a strong premise on which future implementation using green sensors, automatic cloud transfers, advanced data handling, and refined graphics can be utilized in order to improve usefulness and clinical understanding.

CHAPTER 1
INTRODUCTION

In this chapter, the introduction to the IoT-enabled healthcare systems with a particular emphasis on remote patient monitoring (RPM) through hybrid edge-cloud constraints is introduced. Additionally, it defines the purpose of this thesis, highlights the major contributions and provides the summary of the system architecture and technical operational environment of the implementation.

1.1 Background

The current trends in the development of Internet of Things (IoT) technologies have produced a significant impact on the sphere of healthcare, especially Remote Patient Monitoring (RPM). These are systems that rely on smart, networked devices to keep current records of the vital signs so that clinicians & paramedics can provide patients with chronic conditions like diabetes, cardiovascular diseases and respiratory disorders with timely and proactive care [1].

The implementation of a hybrid solution based on the combination of IoT sensors, edge computing, and cloud infrastructure is one of the available opportunities. The patient data including but not limited to glucose levels, body temperature and heart rate are gathered in this arrangement using smart sensors and first processed using a local device, like a Raspberry Pi. The local execution of time-sensitive operations, such as anomaly detection or threshold checks, is carried out to make such tasks respond quickly without being dependent on the cloud latency [2].

To reinforce the same forecasting models like SARIMAX and LSTM are run on Raspberry Pi. These models provide both short-term projections and explain the trends in the data of the patient, which can further be presented directly into the displays of clinicians on linked screens or on-site devices [3]. Therefore, healthcare professionals can receive real-time feedback and intervene before the patient defined parameters crosses certain threshold values to dangerous levels.

On the other hand, information or data that is very crucial in terms of time like summary reports of short time spans and other critical historical parameters related data are sent to the cloud. Cloud-based services provide secure storage, scales and remote access, thus providing healthcare professionals the opportunity to analyze patient data and predict results through AWS dashboards through laptop or smartphone [4].

AWS Lambda and the Simple Notification Service (SNS) is also a part of the system that is integrated with automated alerting. In case any of the monitored parameters goes beyond a preset limit, Lambda triggers an SNS notification that is immediately sent to healthcare personnel through a notification on their SMS, email, or an app [5]. This system ensures that the emergencies are met within the shortest time possible regardless of the location of the medical staff.

However, the use of IoT-based healthcare systems casts the major concerns of data privacy and security. Protecting sensitive health data with strong security encryption method as well as secure authentication against unauthorized access is required in the local processing and network transmission [6].

Although more complex methods of privacy-preserving analytics are becoming available like homomorphic encryption, the standard solution that is used in the present case is through established standard. The system uses the Advanced Encryption Standard (AES) that encrypts the sensitive data before sending it to the network layer (MQTT/TLS). The microcontroller that is Raspberry Pi in this research encrypts the data using AES which is a widely used solution as well as quiet reliable .[7].

1.2 Problem Statement

Remote monitor management or Remote Patient Monitoring (RPMs) systems that have faced serious challenges related to patient disengagement as far as internet-of-things (IoT)-based devices are concerned that connect to the cloud to monitor the health outcomes of the patients has led to significant population in the monitoring of patient outcomes, albeit remotely[8]. Hybrid edge/cloud systems are used to enable the local processing of the most sensitive physiological data on packaged platforms such as the Raspberry Pi, and less time-intensive data sent to cloud systems to store it longitudinally, process it externally, and make it more accessible. However, there are much more technical and operational challenges to overcome in order to prove the reliable implementation of such systems in clinical environments of the real world [9].

The major limitation is the lack of proactive approaches to clinical intervention through the predicting capability. Traditional RPM systems are usually use reactive approach because when they provide notifications after a health anomaly has already happened. In comparison, predicting a wide range of physiological parameters, including heart rate and blood glucose concentration, with the help of the

time series models, e.g. the SARIMAX, provides a significant clinical value due to the ability to identify and predict the trends and anomalies before reaching critical levels. Based on this, real-time alerts based on these forecasts have the potential of improving medical responsiveness and reducing the risk of patients [10].

Moreover, automated alert mechanisms, which are responsive as well as scalable are strongly required. The incorporation of the cloud-based infrastructures, including server less functions and notification structures, is now mandatory to ensure instantaneous communication with caregivers whenever forecasted value thresholds are estimated to be overstepped [11].

Patient confidentiality and data security remain to be the most critical issues. Health information also makes it vulnerable to interception and misuse as it spreads across network infrastructures so therefore, a strong encryption scheme, e.g. Advanced Encryption Standard (AES) protocols, is necessary to protect sensitive confidential information [12].

The current research meets these obstacles by determining a simulated hybrid IoT-cloud RPM framework that assists in secure remote monitoring of health, predictive, and automated. The suggested solution was tested in a controlled environmental experience thus creating a base along which future practical applications can be implemented in clinical practice [13].

RPM systems require constant information flow between the IoT devices, edge processors and cloud assemblies. However, purely cloud-based solutions are likely to cause delays, network failures, or service failures that may disrupt the process of monitoring and delay key interventions. To alleviate these limitations, the system architecture will implement a fault-tolerant approach whereby vital processing will be done locally on a Raspberry Pi, and the non-critical data uploaded to the cloud. This hybrid system ensures that real-time checking and prediction are maintained during any cloud failures or network limitations, which make the system more dedicated and responsible to unpredictability in the operation conditions [14].

1.3 Research Motivation

1.3.1 Enhancing Real-Time Healthcare Monitoring

In the modern era Hybrid Remote Patient Monitoring (RPM) systems need has been increased combining edge and cloud computing to ensure a significant decrease in the latency as well reducing the reliance on internet access. This paper will resolve these goals using the SARIMAX and LSTM forecasting as some of the key data processing tools running on local hardware processors, specifically Raspberry controllers, allowing real-time monitoring and early prediction of health abnormalities [15].

1.3.2 Localized Processing for Improved Performance

Traditional cloud-based frameworks are also limited as they depend on constant internet connectivity which is especially limiting in remote locations. By incorporating localized processing, the application of SARIMAX and LSTM forecasting on local microcontroller, this paper targets at processing of important patient's health information even when there is no connection to the cloud and, thus, improve the quality and real-time responsiveness of healthcare systems [16].

1.3.3 Securing Sensitive Patient Data

Since the security of patient health data is increasingly becoming vulnerable to cyber-attacks during transmission, there is the need to protect vital information. The current contribution focuses on the implementation of powerful encryption algorithms and secure transmission units to ensure data security in the process of local processing in edge devices and the later conveyance to cloud systems to avoid breaching data security regulations and maintaining the confidentiality of the patient information [17].

1.3.4 Developing Fail-Safe and Fault-Tolerant Systems

The need to have the healthcare systems to be operational during removal and provisions of medicine equipment or hiccups in the network is one of the reasons why the study of fault-tolerant architecture has been explored. This study enhances the creation of a system that can support ongoing monitoring of patients by applying both local SARIMAX and LSTM-based prognostication, which can allow healthcare providers to make pre-emptive decisions even in the presence of impaired network or cloud services [18].

1.3.5 Optimizing Scalable Healthcare Solutions

The fact that the hybrid systems can offer scalable and efficient healthcare services promotes the search to seek optimized solutions to process the critical and non-critical health data. The study, by taking advantage of timekeeping forecasting (say using SARIMAX) by means of edge computing and data storage through cloud solutions and other non-critical analytics, attempts to come up with a balanced, efficient, and scalable health architecture [19].

1.4 Research Objectives

The study aims at coming up with a hybrid Internet of Things (IoT) and cloud based Remote Patient Monitoring (RPM) system which aims at ensuring secure, reliable, and real time processes of health data. The goals are defined to offer a methodology that will be followed systematically to find solution to system performance, system resilience, and patient data security in both connection and disconnection conditions concerning partial and fully disconnected operational settings.

1.4.1 Develop a Resilient Edge-Cloud Coordination Framework

The vision is to design an intelligent data-handling model that will need less of frequent connectivity to the cloud. Such structure would guarantee the activation of vital signs and other important physiological parameters that should be processed continuously on the edge, thus assisting the process of making clinical decisions in real time under conditions where the network access is unstable or not available.

1.4.2 Improve System Reliability & Continuity

The task is to adopt system-level measures that ensure continuous monitoring and triggering alarm through a hybrid edge-cloud monitoring. The latter includes the architecture of modern processing pipelines, which are (local) edge-computed and cloud-computed, the inclusion of redundant workflows, and the implementation of operational reserves that maintain continuous patient attention during the occurrence of partial system failures or disconnections.

1.4.3 Build a Fault-Tolerant and Self-Recovering Architecture

The project aims at designing a set up that can dynamically recycle failure of edge devices or cloud services. The intelligent error detection, robust recovery mechanisms and adaptive workload

distribution will be supported by the system and hence the system will ensure that health-care delivery is not compromised in case of faults.

1.4.4 Integrate Forecasting Capabilities to Go Ahead & Provide Proactive Care

It aims to deploy a forecasting model that is based on SARIMAX at the edge to make predictions regarding patient health parameters. This allows identifying the aberrant tendencies in the indicators of heart rate and blood sugar at an early stage and providing the relevant medical care in time, preventing the occurrence of the health emergency.

1.4.5 Enhance Data Privacy and Transmission Security

The focus is to strengthen the data privacy and transmission security past the edge node by incorporating superior security protocols that are able to protect patient health information when communicating between edge devices and cloud solutions. This entails the introduction of the modern encryption policies, safety transport layers and adherence to the health care data policies to ensure the privacy and integrity of data after leaving the local processing environment.

1.5 Main Contributions

This paper outlines the design and implementation of a hybrid remote patient monitoring system using Python based forecasting models to predict blood glucose and heart rate measurements.

1.5.1 Architecture and Implementation of a Hybrid RPM Architecture

The system was produced as a real-time remote patient monitoring system that consists of Internet of Things (IoT) sensors, local microcontroller and cloud services, thus, providing efficient data processing and extensive accessibility.

1.5.2 SARIMAX & LSTM Models Based Forecasting

In order to support local time-series analysis of the health trends and be able to detect anomaly early on and, at the same time, without relying on constant connectivity to the cloud, the SARIMAX and LSTM models were optimized in the Raspberry Pi microcontroller on the edge layer.

1.5.3 AWS Accessibility Through the Cloud

The Amazon Web Services (AWS) S3 is used to facilitate remote accessibility. Patient data with predicting patient outcome are posted into an S3 bucket, where they are safeguarded and can be

accessed at any place. This structure will allow clinicians to access and review information remotely using laptops or mobile devices hence helps in convenient monitoring of information and long-term availability of longitudinal information. Moreover, the cloud can be used as a backup facility in case the local processing or hardware failure takes place, data on critical patient information will still be accessible through the AWS environment, allowing discontinuous monitoring.

1.5.4 Framework of Edge-Cloud Data Management

A dual-layered architecture was also used whereby vital information will be processed at the local location but will also be sent to the cloud to be analyzed and stored in a non-critical manner.

1.5.5 Improved Data Protection and Privacy System

To ensure protection of sensitive health-related information during both local and cloud communication, encryption protocols and other secure transmission systems were established in line with the modern cybersecurity guidelines.

1.5.6 Fault-tolerant and Reliable System Operation

The system design was made to accommodate failover management and the ongoing use during the cases of internet or cloud failures hence assuming strength in real-world clinical and remote-monitoring environments.

1.6 Scope of Study

This research focuses on the design, simulation, and validation of a hybrid edge-cloud architecture for remote patient monitoring (RPM) using IoT technologies, with an emphasis on real-time forecasting, automated alerting, and data security. The scope includes the development of a prototype system capable of simulating real-time acquisition, forecasting, and secure transmission of vital signs such as heart rate and blood glucose levels.

The system architecture is implemented using a Raspberry Pi microcontroller as an edge device to process the data locally at the edge of the network, SARIMAX as well as LSTM models for time-series forecasting, and different AWS services for cloud-based alert generation. To enhance usability, and ensure user friendly atmosphere a local web-based controller is also integrated, providing a dashboard for stats, visualization trends, and graphical display of patient data. Whilepatient data is colleted from different platform including kaggle and john Hopkinson , the system is designed in such

a way to implement real sensor integration in future stages. Security is addressed through the implementation of AES encryption as well as JWT to protect patient data during transmission and storage.

The study is aligned with several **United Nations Sustainable Development Goals (UNSDGs)**:

1.6.1 SDG 3 (Good Health and Well-being)

The system allows proactive health data monitoring by offering automated alerts and real time monitoring, so by supporting timely medical intervention. It reduces the load on healthcare facilities and health care professionals thus by improving outcomes for individuals with chronic conditions, regardless of their location.

1.6.2 SDG 9 (Industry, Innovation and Infrastructure)

In modern health care by means of integrating IoT, edge computing, and cloud services, this RPM system produces innovation. It boosts the adoption of smart technologies in medical, enhancing efficiency and encouraging scalable health solutions not only in urban but rural areas as well.

1.6.3 SDG 10 (Reduced Inequalities)

The system minimizes disparities in healthcare access by providing remote monitoring tools that performs in areas having limited medical infrastructure. It confirms that patients from diverse socioeconomic and geographical backgrounds receive best possible health services, rather than physical proximity to hospitals.

1.6.4 SDG 11 (Sustainable Cities and Communities)

Early detection and quick response to health anomalies on the basis of patient data, enables the system to keep the community safe stronger. It optimizes emergency healthcare response, reducing hospital congestion and improving patient outcomes during critical incidents.

The scope does not include hospital deployment, long-term clinical trials, or integration with national healthcare databases or electronic health record (EHR) systems. These aspects are considered future

1.7 Used Environment

The following tools and environment supported data processing and visualization during the whole phase of the simulation.

1.7.1 Visual Code Studio with Python

The main programming language used in the development of the forecasting scripts and the creation of synthetic datasets were based on Python. Every source code was then copied to code and then subjected to strict testing and debugging using the Visual Studio Code, which provided the relaxed extension capability, interactive debugging and property integration with the necessary libraries. This modeling step ensured that the models were valid before the actual deployment.

1.7.2 Forecasting Models (SARIMAX and LSTM)

Both SARIMAX and LSTM models were constructed to make a prediction of the heart rate and level of blood glucose using time-series. To compare the accuracy and reliability of the models, the models were trained and tested using simulated datasets. The performance metrics that were created then informed the choice of the best methodology to be used in real-time deployment.

1.7.3 Raspberry Pi 5 Deployment

After all python scripts were tested and passed in the desktop environment, they were moved to a Raspberry and ran on the Raspberry 5. The tool was a form of an edge-computing platform that executed the forecasting models in real time and at the same time handled information transmission between itself and other systems.

1.7.4 Local Web-Based Controller and Dashboard

The Raspberry5 was built as a Modbus TCP/IP server, which allowed an easy connection to a locally available web-based controller. The Raspberry was automatically recognized by the controller as a device connected to it and the parameters was recovered. With the help of a user-friendly dashboard that works with the help of a browser, the real-time data, the visual trend displays, and the log files were accessible, thus, allowing the users to remain connected even when there is no connection to the cloud.

1.7.5 AWS Cloud Integration

In the case of secondary and distal access, the system outputs were posted into an AWS S3 bucket, thus ensuring a safe storage and access of the clinicians through cloud services when

there is a local hardware or network disruption. AWS was therefore, an effective reserve and visualization system that supplemented the local edge infrastructure.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

The chapter covers the literature review on healthcare monitoring systems based on IoT, sensor integration, edge-cloud architectures, physiological parameter forecasting can be described as well as alert mechanisms using cloud environments. It explores the acquisition of sensor measurements, such as heart rate and blood glucose, through embedded systems as well as processing them on local edge computers, such as the raspberry pi 5 microcontrollers, to provide real-time response. In addition to that, it investigates the use of cloud services, as in the case of Amazon Web Services (AWS), when storing data long-term, making predictions, and ensuring remote access. The paper also discusses how machine-learning models like SARIMAX and Long Short-Term memory (LSTM) networks, can be used to make predictions of time-series and how AWS Lambda functions can be used to provide an automated mechanism of alert generation. Besides, the literature identifies issues with data interoperability, security, and effective management of large sized medical dataset thereby, contributing to the design choices and study direction of this project.

2.2 Transforming Healthcare through Remote Patient Monitoring

Remote Patient Monitoring (RPM) is a revolutionary development in the health-care sector which applies digital technologies to support the purchase, processing, and management of patient information that is generated in geographically distant locations. RPM has significantly improved patient outcomes, especially among underserved segment, in the last ten years in developed countries. Early applications of RPM used mainly manual data delivery to primary vital signs including blood pressure and heart rate due to its initial limitation to the monitoring of these parameters. The next phase of integration of tele medical systems with advanced data analytics pipelines has pushed RPM into a more intelligent, responsive and proactive paradigm. However, despite its potential, it is still complicated by the fact that the process of translating RPM into the regular clinical practice remains constrained by the issues of system scalability, data integration, and flexibility across the heterogeneous clinical conditions [20].

RPM grants a plethora of advantages to the quality of delivery of health care. Continuous monitoring provides clinicians with the near-real-time data, thus, making intervention decisions faster and

decreasing the readmission rates and eventually perfecting clinical decision-making processes [21]. Furthermore, RPM enables patient empowerment, which is the promotion of the participation of the patient, in patient care with further enhancement of treatment adherence and clinical outcomes [22]. At the same time, it helps to contain the costs by reducing unnecessary hospital visits and making it possible to detect any health-related issues early [23]. These benefits are especially relevant to the treatment of chronic diseases like diabetes mellitus and chronic obstructive pulmonary disease (COPD) when real-time data can be used to make timely practice changes [24]. Nevertheless, these advantages mainly depend on the dependability of communication of data and the competency of the end-user to work with technology, which cannot be the same between different groups of people and other healthcare systems.

In the post-discharge scenario, RPM enables measuring patient recovery, which contributes to reducing the readmission rate and complications earlier in the progress of their development [25]. However, there is an ongoing gap with respect to customization of modern RPM systems to suit heterogeneous patient needs. Many systems are not very adaptable in the occurrence of comorbidities thereby limiting their ability to generalize across wide clinical settings.

The fact that RPM has spilled over into the sphere of non-communicable diseases (NCDs) enhances its strategic importance. With the help of progressive interoperability with Electronic Health Records (EHRs), information about a patient can be shared between interdisciplinary care teams, thus, improving service coordination. With recent developments of new modalities, including mobile health applications and wearable sensors, the abilities of RPM have only increased to make monitoring more continuous and individual based. However, the literature available on the long-term performance and durability of these integrations is still weak, particularly in the context of low-resource settings where the adoption of EHR remains a disjointed event.

The RPM has more been contemplated as the tool to reduce health disparities in resource-lad settings. It provides a channel to have access to healthcare in places where face-to-face meetings are not practical through the implementation of mobile and wearable technologies. However, the promise is constrained by the lack of digital literacy, insufficient infrastructure, and interruptions of the internet connection. Without some remedial measures to be taken on these underlying barriers, RPM is likely to support rather than alleviate existing inequities.

RPM still suffers critical technical and systemic barriers to its implementation despite its promise. Information security is one of the major issues, and it is extremely important to comply with such regulatory standards as HIPAA and GDPR. The use of cryptographic protocols, such as Advanced Encryption Standard (AES) and JSON Web Token (JWT) authentication, can be used to strengthen the security configuration [26], but it is not evenly distributed over heterogeneous platforms. The lack of interoperability also affects the device level tremendously, the absence of universal standards may promote the fragmented care and lack of quality data sharing. Integration [27] cannot be smoothly implemented without the adoption of standardized protocols such as HL7 and FHIR. In addition, areas that do not have solid internet buildings have reduced system reliability, thus restricting the real-time responsiveness of RPM solutions [28]. The flood of non-filtered information provided by many patients also strains clinicians, and in order to reduce information overload, smart data triage systems are required [29].

Finally, although RPM has potentially transformational power, critical evaluation demonstrates its benefits to be pegged on successful addressing of salient technical, infrastructural and socio-economic issues. The subsequent research ought to focus not only on the technological sophistication advancement, but also on working out the comprehensive and adaptable frameworks that will make RPM equitable and sustainable in the varying healthcare milieus.

2.3 Hybrid Edge-Cloud Architecture in RPM

The integration of edge computing and cloud services has become a stable framework in Remote Patient Monitoring (RPM) enable the user to avail the benefits of both local data processing and centralized data management. This hybrid architecture aims to overcome the limitations of the all-systems that are purely cloud based by installing smart edge nodes closer to the patient and at the same time uses cloud systems to store, analyze, and access information remotely. However, the strategy is bright, but in real practice, it has certain challenges in the form of the complexity of the system, its cost, and synchronization of data between distributed parts.

2.3.1 Benefits of Edge Computing in Healthcare

Edge computing in RPM provides an opportunity to process health data at the local level, which lowers the latency considerably and improves system responsiveness. This has been crucial at the time of emergencies where emergency intervention is required including cardiac arrest cases or respiratory

failure [30]. In addition, edge computing can reduce the amount of bandwidth used to transmit data and enhance data privacy because sensitive data do not necessarily have to be sent to remote servers in their entirety [31]. However, even though this model supplements the performance in real time, it has restriction, and especially the ability of edge devices to handle complex analytical models or implement long-term data storage. In addition, there are still important issues of maintenance and fault tolerance of edge nodes deployed in remote environments [32].

2.3.2 Role of Raspberry Pi 5 as Edge Node

Raspberry Pi 5 is progressively being used as an edge computing node as its hardware features have been enhanced making it cost-effective and flexible. It also facilitates local data collection and light inference in the early detection of anomalies and, therefore, is appropriate in RPM systems [33]. Nevertheless, even with such benefits, Raspberry Pi 5 devices are not necessarily mission-critical medical devices. They are not as reliable as industrial quality, and might have hardware constraints when put under extended load, or under resource-bounded deployments. Although the ability to be compatible with various medical grade sensors is one of the strengths, the ability to maintain the real-time functionality in varying operational environments deserves additional consideration [34].

2.3.3 Cloud Computing in Healthcare (AWS Focus)

Cloud platforms like AWS offer scalable infrastructure and HIPAA-compliant environments to support RPM system needs. Services such as AWS Lambda, S3, and SNS allow for remote data access, alert automation, and robust storage mechanisms [35]. Additionally, the integration of AWS dashboards enables physicians to visualize trends and access predictive models, contributing to informed medical decisions [36]. However, critical analysis reveals potential drawbacks in cloud dependency, such as reliance on constant internet access, data transfer delays, and privacy concerns when dealing with sensitive health records. Although cloud providers implement security protocols, data breach risks and compliance overhead remain pressing concerns in healthcare IT ecosystems.

2.3.4 Comparative Analysis Between Fully Cloud & Hybrid Models

High latency and bandwidth dependency of cloud-based RPM model impedes fully cloud-based RPM model benefits: central control and simplified management; furthermore, rural areas or locations with limited bandwidth [37]. Meanwhile, the hybrid edge-cloud software allocates processing resources intelligently, performing real-time operations locally and transferring complicated computations and

archiving functions to the cloud [38]. This design promotes responsiveness and resiliency of the system. However, the dual-layer architecture introduces more complexities into the system and requires powerful synchronizations and the coordination of error management. Lack of proper orchestration also can create data inconsistencies or system bottleneck, therefore compromising the reliability of monitoring patients [39].

2.4 Role of Time Series Forecasting in Healthcare

Timely prediction of patient vital signs is an objective constituent of predictive healthcare systems, which offers a preventive model of therapeutic intervention and emergency management. When statistically processing the stream of physiological data, the clinicians are enabled to determine early warning signs and avoid the occurrence of preventable complications. However, as much as predictive analytics bring with them significant benefits, methods and settings that are used to apply these models have a significant impact on the performance of the system, its level of complexity, and its real-time capability.

2.4.1 Significance of Forecasting Vitals

The ability to predict such key physiological parameters as heart rate, blood glucose level, and body temperature allows identifying any abnormalities early on and providing the necessary medical care in a timely manner, in particular, in high-risk groups of chronically or older patients [40]. The approach ensures continuity to care and helps to sustain remote medical supervision [41]. Yet, predictive insights can also be used to supplement decision-making, but their usefulness is very dependent upon the standards, depth, and uniformity of input data. Independent measurements made by a sensor or disruption of data can provide inaccurate forecasts, because of which clinical judgments are also a significant threat [42]. Accordingly, effective forecasting requires an industrial predictive model as well as good quality and well-read data acquisition mechanisms.

2.4.2 Summary of Statistical Models

Historically, traditional statistical models, such as autoregressive (AR), moving average (MA), and ARIMA models, have been used in the analysis of healthcare time-series because of its mathematical simplicity and interpretation [43]. Even though these algorithms are simple to apply, they often have poor performance in dynamic, noisy environments where any exogenous factors- drug changes or environmental changes- influence physiology measurements [44]. Advanced methods which include

seasonal ARIMA (SARIMA) and SARIMA (exogenous) address at least some of these weaknesses and therefore introduce seasonality and exogenous variables, which boost predictive characteristics and predictive reliability [45]. However, critical reflection shows that at least these sophisticated models can be hindered by the inabilities to reproduce nonlinearities, and tuning of the parameters will be needed, and it may be limiting to the scalability and effectiveness of these models within real-life, multi-patient monitoring conditions.

2.4.3 SARIMAX Model

SARIMAX model complements prediction accuracy by incorporating seasonal variations and exogenous covariates such as medication regimes and ambient temperature that are commonly used in the inpatient closely monitored setting [46]. Its main strength is in that it is interpretable and expects a linear modeling such that it makes it accessible to clinicians who otherwise might not be that familiar with more complex machine-learning paradigms. In addition, SARIMAX models are also usable on underprivileged hardware like the Raspberry, making it possible to perform edge-based real-time prediction and reduce the need to use cloud networks [47]. This methodology however assumes relatively clean and stationary data. In reality, patient data can be irregularly sampled, non-stationary and noisy and this can cause a diminution of the model performance without proper pre-processing and periodic recalibration. Furthermore, running SARIMAX on the edge devices will require lightweight optimization and shall create a computational load, which will weigh off against the processing power of individual devices.

2.4.4 LSTM Model

Long short-term memory (LSTM) networks are a special type of recurrent neural network, which is designed in a way that it allows long-range dependencies to be captured in sequence data. They are therefore especially effective at predicting patient health outcomes of the form that antecedent measurements have delayed impacts, including those of postprandial glycemic reactions or the cumulative impacts of drugs. In contrast to SARIMAX, which is limited to linear, seasonal models, LSTM networks have the ability to believe complicated nonlinear dynamics and abstract temporal interactions without requiring data requirements (they can be thought of as hierarchical models with one layer). The empirical evidence has shown that LSTM models outperform SARIMAX in time-series forecasting activities, particularly when more complex and nonlinear healthcare signals are used [48].

Regardless of the above capabilities, LSTM architectures have very high computational requirements, and require very large training sets, making them difficult to run on small hardware platforms such as the Raspberry Pi. In addition, they are too opaque, i.e. black-box in nature, which limits interpretability, so clinicians have less transparency compared to traditional models. However, the LSTM networks can provide a high level of predictive accuracy and can be used as an important tool in an application of continuous monitoring of patients in cases when enough data and computer capacity are at their disposal.

2.5 Alert Generation and Dashboard Integration

The generation of alerts and data visualization, which are user-friendly is a key requirement before translation of the raw sensor outputs into form of actionable healthcare intelligence. The current section examines the use of Amazon Web Services (AWS) in order to automate alerting, as well as analyzes the role of dashboards as providers of real-time and prognostic information on the basis of which clinical decisions can be made. There is a critical evaluation in relation to such issues as scalability, responsiveness and interface limitation.

2.5.1 AWS Lambda for Event-Driven Automation

AWS Lambda is used to run stateless functions in response to discrete events in healthcare, specifically anomalous sensor reads or trigger overage. It is highly responsive almost in real time and therefore it is ideal when it comes to time sensitive remote patient monitoring (RPM) use [70]. However, both the execution time constraints inherent with Lambda, and its cold-start latency, is a constraint to systems with real-time performance needs that must run continuously. Moreover, the coordination of several Lambda functions to effect event sequences of large healthcare processes magnifies integration complexity.

2.5.2 AWS SNS for Health Alerts

Amazon Simple Notification Service (SNS) will facilitate rapid distribution of notifications instead of abnormal health parameters in real time via the use of notifications via SMS, email, and push [49]. This feature will ensure that inter-communication between patients, caregivers, and clinicians is timely. Even though SNS enhances scalability and operational simplicity the effectiveness depends on the network availability and latency of third-party messaging providers. In rural or bandwidth-limited settings, the ability of delayed alert transmission to obstruct timely clinical response can be important

and it follows that as a safety measure, a local alert redundancy and hybrid fallback architectures are needed.

2.5.3 Visualization Dashboards (AWS, Alternatives)

AWS Cloud Watch and Grafana are visualization tools that provide dynamic dashboards to visualize raw and processed health data in intuitive charts and graphs [50]. These platforms, which are used in the field of trend analysis and anomaly detection, will allow making evidence-based clinical decisions. However, dashboards are often configured by hand and do not need to be understandable to other healthcare professionals that are not technical. Moreover, the real-time performance of many concurrent users over large scale deployments needs to be consistent with optimization of the back ends and the robust data-streaming data structures.

2.5.4 Real-time vs. Forecasted Data Visualization

A combination of real-time data streams and SARIMAX forecasts on combined dashboards allows clinicians to compare real-time data with forecasts on the same diagrams [51]. This kind of integration facilitates proactive care and therefore it is possible to intervene before adverse events occur. However, there is a problem of not discerning between real-time variability and forecast uncertainty when there is no conclusive visual indication or confidence range. Misunderstanding the forecasting information can initiate an event of false alarms or missed interventions, which explains why a careful design of the visual encodings as well as interpretability rules that suit the needs of the medical experts is the key element.

2.6 Data Security in Healthcare IoT

The privacy of sensitive health information is of utmost concern in Remote Patient Monitoring (RPM) systems due to the high-probability of cyber-attacks, intrusions, and information breaches [52]. These systems often work in a distributed environment that contains heterogeneous devices and it makes the end-to-end security interesting and difficult to assure in a technical and operational manner.

Data transmission without encryption, insecure authentication protocols, vulnerable application programming interfaces (APIs), are some of the common vulnerabilities that can be used to compromise the confidentiality of patient information [53]. The physical and network-layer threats may frequently affect edge devices, including Raspberry Pi, when no measures are taken to protect such devices, people can use them as entry points to malicious networks. Furthermore, the cloud

endpoints, although scalable, bring on-board additional risks as they have a shared infrastructure and may face a multi- tenant issue.

To alleviate these problems, there is the usage of the Advanced Encryption Standard (AES), especially the AES-256 to encrypt all sent and stored health information [54]. Despite the fact that AES is associated with a high level of security, it might need to be carefully optimized when deployed on resource-limited devices to avoid a performance loss. There should be trade-off of encryption strength and processing efficiency that should be questioned in real-time RPM settings to avoid jeopardizing the data confidentiality without limiting low-latency responsiveness.

JSON Web Tokens (JWT) support stateless authentication, which can be used to carry out secure session management and user authentication over distributed systems [55]. However, unless properly administered, JWTs can be susceptible to token hijacking and abuse in the same way that they can be in an environment not using a secure token storage or expiration time. This is why their incorporation must include token renewal plans and means of secure keys distribution to support the integrity in authentication.

Adherence to international data protection regulations, including HIPAA and GDPR, is essential when it comes to being legally responsible and morally responsible in their data management practices [56]. Nevertheless, the alignment of Internet of Things (IoT) systems to these standards is a complex one, which requires constant updates, monitoring, and auditing. Smaller providers in the healthcare settings might experience limitation of resources that may not allow complete compliance which may be a challenge in the implementation of secure RPM solutions.

Finally, although the integration of encryption and authentication solutions enhances the security posture of the IoT-based healthcare infrastructure, a fatal dissonance remains in the process of balancing between usability, regulatory compliance, and performance [57]. The future systems will have to integrate adaptable security models that enhance security response to emerging threats and expansion of the system to enable patient trust and secure scalable solutions.

2.7 Current Challenges and Research Opportunities in RPM Framework

Despite the relevant progress in IoT-based Remote Patient Monitoring (RPM) and automation in the healthcare domain, there are still quite a few unresolved issues that hinder their applications in realistic scenarios. These issues cut across architectural, operational and regulatory levels thus creating significant research gaps that require further emphasis and innovative research.

On the edge of the network, real-time data processing is an essential constraint, particularly where applications, e.g. latency-sensitive ones, need to run on limited computational resources. The fact that edge nodes like Raspberry Pi have been enhanced does not mean that they can generate local analytics correctly due to hardware limitations. The resulting need is in hybrid architectures, although there is a complexity of coordination of edge and cloud layers, which introduce areas of failures especially when the network becomes disrupted.

Heterogeneity the Eastern communication protocols and the variety of medical devices raise the problems of scalability and interoperability. Although there are standardization efforts, e.g. HL7 and FHIR, their use has not been uniformed amongst different vendors, which represents a bottleneck to integration and data silos. Such fragmentation makes aggregation of health-data and cooperative care impossible, which are required to monitor the patient comprehensively.

RPM systems that have been integrated with predictive models like the SARIMAX make it even more complex. Whereas such models render more advanced approaches to proactive care, with predicting anomalies, their real-time implementation in edge devices is limited by processing overhead and the differences in data. Whereas, tuning of SARIMAX hyper parameters and making it robust in non-homogenous datasets in the health sector is not a straightforward task, compared to deep-learning models, this model is relatively lightweight. Additionally, accuracy of forecasting may be reduced where there are noisy or sparse sets of data, which restricts the feasibility of such techniques in critical-care practice modes.

Security wise, the protection of data in transit and rest has remained an unsolved issue. The attack surface is expanded by the integration of various platforms and services, the sensors to cloud analytics. Although some RPM systems may use encryption and secure protocols, due to the lack of real-time threat detection and adaptive security policies, most of the systems are susceptible to zero-day attacks as well as leakage of data.

When hybrid edge-cloud RPM architectures are deployed in environments under-resourced or with high latency this will reveal critical infrastructural constraints. Without the provision of good power supply, strong internet connectivity, and technical support, such solutions stand the risk of becoming unsustainable. These environmental conditions therefore should be taken into consideration in the design and testing stages of RPM systems in order to have a fair access to healthcare.

There are, thus, direct research imperatives in the refinement of fault-tolerance approaches, the creation of intraoperative models, the optimization of predictive models on the deployment of edges, and the instantiating of adaptive measures on cybersecurity. This paper deals with these aspects by presenting an opportunity to come up with a secure, predictive and resource-conscious hybrid RPM architecture that has the ability to work efficiently in various and demanding healthcare conditions.

Table 2.1: Research Gaps and Challenges

Focus Area	Drawbacks Identified	Research Gaps	Reference
Cloud-IoT Integration for Healthcare	Limited data visibility, dependency on reliable internet	Need for real-time processing at scale, improved security frameworks	[58]
AWS Server less Architecture in Healthcare IoT	Managing high-velocity data streams, lack of flexibility in AWS Lambda	Optimization of server less solutions for dynamic healthcare needs	[59]
Chronic disease management through continuous monitoring and remote accessibility.	Possible delays in real-time decision-making due to data processing distribution.	Balancing real-time analysis and scalability for chronic care.	[60]
Scalability and cost-effectiveness in underserved regions.	Limited healthcare infrastructure and network availability.	Developing solutions for regions with poor connectivity.	[61]
Data security and encryption for transmitting sensitive health data.	Vulnerability to breaches during data transmission.	Implementing advanced encryption and secure communication protocols.	[62]
Interoperability among devices and platforms from different vendors.	Lack of standardization hinders system integration.	Designing universally compatible and interoperable platforms.	[63]
Efficient management of large-scale IoT data and analytics.	High data volume challenges real-time processing and storage.	Improving data management strategies for hybrid RPM systems.	[64]
Dynamic Model Selection for Time-Series Forecasting	SARIMAX low accuracy on long data, LSTM low accuracy or high data requirement for short data. Selection depends on data volume.	Benchmark SARIMAX (short-term) vs. LSTM (long-term) to justify optimal model application based on data volume for maximum accuracy.	[65]

CHAPTER 3
METHODOLOGICAL FRAMEWORK AND SYSTEM ARCHITECTURE

3.1 Introduction

The chapter presents a map of the methodology and architecture behind the proposed hybrid Remote Patient Monitoring (RPM) architecture, that is in fact a hybrid natured edge-based forecasting system with cloud-facilitated alerting and data management. The system is designed to enhance real-time health tracking with the help of predictive analytics, automated notifications, and scalable storage systems. Continuous Glucose Monitoring (CGM) sensors form the main source of physiological data, however in the current implementation artificial patient measurements of blood glucose are recreated through Python scripts which are run in Visual Studio code. As a result of the above, the suggested framework can be tested and confirmed within the limits of controlled experiments and then implemented with the real medical equipment.

In order to get a high level of short-term forecasting, two time-series forecasting algorithms SARIMAX and LSTM models are also executed on a Raspberry Pi 5 that serves as an edge controller. The layer of the edges ensures that it adds to the rapid processing of the abnormal physiological patterns characterized by low-latency processes that do not have to solely discuss the cloud service. Standard statistical measures of accuracy Mean Absolute Error named MAE and Root Mean square error named RMSE are used to scrutinize the outputs to measure the performance and reliability of the model. The combination of these models allows the system to make sure that individual linear seasonal trends, as well as complex nonlinear variations in patient data, are properly represented and predictive insights obtained are stronger.

In addition to edge-level computation, there is remote accessibility that is achieved via a two-way solution, a local web-based dashboard created using a local web server based controller and cloud-based integration using AWS services while patient data is processed locally through sensors by using a local microcontroller Raspberry Pi 5 in this particular case. The hybrid access model is flexible and the healthcare providers can monitor the patients both at the on-site interface and remotely on the cloud. AWS Lambda and Simple Notification Service (SNS) are applied in order to automate notification, thus, interfering on time in case of expected abnormal conditions. The general design

promises safety in communication of data, scalability and reliability creating a practical flexible base on patient-centered healthcare monitoring.

3.2 Research Methodology

This segment describes the research approach that was adopted in the development and validation of the proposed hybrid Remote Patient Monitoring (RPM) system which incorporates edge computing and cloud service provision. The application of methodology is, simulation- Based and System design oriented based on the requirements of healthcare monitoring which are specific to the situation of a resource constrained but scalable environment.

3.2.1 System Design and Simulation Overview

The study is aimed at designing and validating a hybrid healthcare monitoring system, which was developed in the context of a practical and applied approach. As opposed to depending on a strictly experimental or survey-based approach, the focus is put on creating and making a working prototype that combines edge-level computing with cloud-based alerting and data management. The aim is to develop a proactive and robust Remote Patient Monitoring (RPM) system that can be used in both practical conditions and under resource settings that are resource-sufficient.

The core of the system is covered by a Raspberry Pi 5 which serves as a local edge controller. It receives data streams of incoming health data, does short term forecasting and operates local-based visualization using web-based server. The Raspberry Pi transfers historical and forecasted information to this local web controller, on which the vital signs and statistical parameters are shown in real time. At the same time, AWS like S3 storage is used to send the same data to the cloud, ensuring redundancy, safe archiving, and external availability. This dual-channel architecture will make sure that healthcare practitioners will have access to patient data locally on site or via AWS-hosted dashboards, based on the need.

Synthetic blood glucose measures of patients in a simulator are created through Python scripts written in Visual Studio Code. These simulated data are the duplicate of a real life input and are coded into CSV files which are used as the basis of training and validating the forecasting models. Both SARIMAX and LSTM algorithms are used to the data and they enable the system to detect the linear seasonal trends and nonlinear fluctuations. After validation, the scripts and the related data using visual

code studio, these scripts are sent to the Raspberry Pi where the models are run within real-time constraints and in essence an actual deployment situation is simulated.

The generated forecasting results are directly connected to automatic alerts. Combinations of predictive insights and threshold-based rules are used to decide when AWS Lambda functions are to be invoked. AWS SNS provides caregivers with notifications through email or SMS once invoked and thus provides timely intervention in the event of abnormal conditions. This integration brings out the capability of the system to be able to integrate predictive modelling as well as automated cloud services in a way that facilitates an end-to-end monitoring process.

In addition to software testing, it is also a practical benchmark to use in assessing the processing capabilities of the edge controller to test the Raspberry Pi deployment. The system not only confirms the accuracy of SARIMAX and LSTM models but also confirms that the Raspberry Pi 5 is capable of functioning as a more than capable but lightweight edge computing unit. This practice demonstration highlights the practicability of implementing this type of hybrid systems into remote locations or scanty resource locations where traditional hospital infrastructure might not exist.

3.2.2 Simulation Justification

The nature of clinical information and the restrictions that are relevant to real-world conditions involving patients leads to the utilization of the simulated environments to test the system is quite essential. Synthetic data of heart rate and blood glucose paths were programmatically created using Python scripts and run in the Visual Studio Code. The artificial variables were developed to reflect the real physiological patterns such as normal oscillations, sudden deviations as well as stochastic anomalies. Such methodology made it possible to test systems comprehensively and inside a controlled, safe environment and maintain phenotypic fidelity to realistic patient dynamics.

Against the simulated datasets two forecasting paradigms were implemented, Seasonal Autoregressive Integrated Moving Average, also with exogenous regressors (SARIMAX) and Long Short-term Memory networks (LSTM). SARIMAX is good at short-term seasonal and linear interaction, and LSTM is good at long-range sequential interaction and non-linear interaction. Through the use of the two methodologies on the same data corpus, the research enabled a strict comparison of the metric of predictive performance and guided the determination of the appropriateness of each model to use as an edge deployment on Raspberry Pi 5 hardware.

System architecture made sure that the results calculated on the Raspberry Pi 5 were also displayed locally using a web-based dashboard and stored to an AWS (S3) bucket. Such a dual-mode approach provided a measure of resilience, there was real-time information in the premises, and the cloud storage acted as a reliable backup, remote access and long-term storage, and redundancy during hardware downturn or the loss in connectivity.

The accuracy of the predictions was measured comparatively and transparently in both Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). These objective performance metrics reported the relative performance of each model when run in homogeneous test conditions so as to cast light on the trade-offs between accuracy and cost of estimation.

Finally, the introduction of the simulation allowed accessing the system over a wide range of conditions without negatively affecting patient safety. There would be exposed to diverse perturbation, e.g., additive noise, anomalous variability, and threshold violation, to generate system responses and support the alerting pipeline based on AWS Lambda and Amazon SNS.

The simulations enabled

- Controlled testing of forecasting models (SARIMAX and LSTM) on identical datasets.
- Direct comparison of predictive accuracy using MAE and RMSE.
- Evaluation of cloud integration through AWS S3 for secure storage and remote access.
- Validation of real-time monitoring via local web-based dashboard.
- Robust testing of the alerting workflow with AWS Lambda and SNS under different health conditions.

3.2.3 Applied Nature and Practical Focus

The approach used is intrinsically applied, because it is based on building functional prototype and assessing it instead of a purely conceptual framework. All the components of the system were tested and python scripts implemented on a Raspberry Pi 5, so it is certain that the architectural design was effective and workable in real-life circumstances. The working nature of the implementation can be observed in the following parts:

- **Raspberry Pi-based IoT data acquisition and processing:** Local processing of synthetic datasets of heart rate and blood glucose using Python Scripts The models used were SARIMAX and LSTM, which served to make predictions.

- **Local visualization through a web dashboard:** Raspberry Pi was set up as a Modbus TCP/IP device to be integrated with Web Controller to access and visualize real time parameters, historical records and graphical trends.
- **Cloud integration with AWS S3:** Data and logs were processed and moved to AWS S3, directly through microcontroller thus ensuring secure data transfer, remote access and has been used to ensure reliable storage in case of local failure.
- **Automation of event handling using the AWS Lambda:** Lambda functions were used to monitor the uploaded datasets and raise an event whenever critical thresholds were exceeded.
- **Real-time notification using AWS SNS:** Alerts were sent to the caregivers or physicians via email and SMS, so timely response was guaranteed in case of an emergency situation.

Such pragmatic concentration depicts that this system was purposefully designed to include low latency edge processing, scalability and resilience with cloud characteristics, and automated notifications, to ensure timely interventions.

3.2.4 General Approach to Building the Hybrid RPM System

The proposed methodology is executed through the following iterative steps

- **Requirement Analysis & Design Planning**
  Definition of system requirements, patient parameters, alert mechanisms, and performance goals.
- **Data Simulation & Preprocessing**
  Generation of mock datasets for heart rate and glucose levels. This includes noise modeling, timestamp generation, and normal vs abnormal scenarios.
- **Edge-Level Implementation**
  - Sensor readings are processed on Raspberry Pi 5
  - Initial filtering and threshold checks occur locally
  - Forecasting using SARIMAX is performed on the edge device for real-time decision-making

3.2.4.1 Cloud Integration

- Alert events are pushed to the AWS Lambda function
- Lambda invokes SNS to send notifications when risk levels exceed predefined thresholds

3.2.4.2 Optimization through Objective Functions

- Risk Index minimization
- Patient-centric alert correctness maximization

3.2.4.3 Performance Evaluation via Simulation

The system is evaluated based on forecasting accuracy, alert precision, latency, and scalability under simulated load.

3.3 System Objectives and Optimization Functions

In the context of remote patient monitoring (RPM), it is essential to define clear system objectives that guide the optimization of care delivery and technological efficiency. The proposed system incorporates two primary objective functions aimed at enhancing patient safety, reducing intervention delays, and optimizing system-level decision-making. These functions form the basis for risk assessment and alert prioritization, ensuring that both clinical and technical goals are met effectively.

3.3.1 Risk Index

Objective: Minimize the health risk index for patients based on predictive trends.

Formula**:**

Risk Index= $\sum_{i=1}^{n}(R_i \cdot \hat{p}_i)$ **Equation -1**

Where:

- R_i: **Risk factor weight** for patient i (Derived from factors such as Medical History , Lifestyle factors and clinical conditions)
- $\hat{p}_i$: **Predicted probability** of adverse events for patient i **(**calculated using predictive models such as regression, machine learning, or statistical trend analysis, which analyze past data to predict future risks.
- $n=$ total number of patients

Example:

Suppose there are three patients so n = 3

Patient "A" $\boldsymbol{R_i}$=0.5, $\widehat{\boldsymbol{p}}_{\boldsymbol{i}}$=0.8 (moderate risk weight, high probability).

Patient "B" $\boldsymbol{R_i}$=0.9, $\widehat{\boldsymbol{p}}_{\boldsymbol{i}}$=0.2 (high risk weight, low probability).

Patient "C" $\boldsymbol{R_i}$=0.3, $\widehat{\boldsymbol{p}}_{\boldsymbol{i}}$=0.9 (low risk weight, high probability)

The Total Risk Index = (0.5 * 0.8) + (0.9 * 0.2) + (0.3 * 0.9) = 0.85

The cumulative risk factor enables proactive management of the system

If the risk is trending upward, medical personnel can adjust system settings, such as lowering alert thresholds or scheduling patient check-ins with much higher frequency.

3.3.2 Patient-Centric Optimization

Objective: Maximize patient outcomes by optimizing alerts and reducing unnecessary interventions.

Formula**:**

Objective Function = max $\{\sum_{i=1}^{n} \omega_i \cdot (Alert\ Corectness_i) - \lambda.(False\ Alert_i)\}$ **Equation -2**

Where:

Alert Correctness $\boldsymbol{i}$ = 1 if the alert for patient I was accurate; otherwise, 0

$\boldsymbol{\omega_i}$ **:** weight given to patient I, shows the importance or priority of getting the alert

λ: Penalty for false alerts to discourage unnecessary interventions.

False $\boldsymbol{Alert_i}$ **:** 1 if the alert was wrong, otherwise 0

In the current objective function, there are two main parameters denoted as ω (omega) and λ (lambda) to control the trade-off between the accuracy and sensitivity of the alert system. The parameter ω represents the relative significance or priority of each patient such that increased ω values are allocated to patients in need of more close attention or prompt warnings. On the other hand, λ is a penalty parameter that dictates false alerts in cases where λ takes a large value, the system takes a conservative approach, which bases its choice on the prevention of an incorrect alert. The metric of alert correctness,

which is an aspect of the indicator, works by tracking the correctness of a previous alert (1) or incorrectness (0).

As an example, the implementation of such weighting factors is considered by analyzing three patients who have different weights of priority. Patient A is given $\omega = 2.0$ and has already created one correct alert and no false alerts and Patient B has $\omega = 1.0$ and has already created one false alert and Patient C has $\omega = 1.5$ and has already created one correct alert. Assuming $\lambda = 0.5$, the objective value is calculated as $[(2.0\times1) + (1.0\times0) + (1.5\times1) - 0.5\times (1)] = 3.0$. This high objective score shows that the system has been able to produce correct alerts to patients with high priorities, on the other hand, reduce false interventions.

3.4 System Overview

The suggested hybrid Remote Patient Monitoring (RPM) system is bound to describe how predictive analytics and cloud-edge can be utilized to manage patient health. In the context in question, the system does not include any direct patient involvement or physical sensors use. Instead, artificial data is created to simulate realistic patient data of parameters that are continuous data. Such datasets are generated in Visual Studio Code by use of Python scripts thus allowing the system to be tested in controlled and repeatable condition without ethical or clinical limitations.

After being created, the generated synthetic datasets are processed in the Raspberry Pi 5 that serves as the edge computing device. The Pi plays a number of important roles at the edge layer:

Preprocessing and filtering of the input data to remove noise and structure the time series.

- Preprocessing and analysis of input data to get rid of noise and organize the time series.
- To better predict the future, it is important to note that there is the use of SARIMAX forecasting that involves the use of exogenous variables.
- Implementation of LSTM models to compare with SARIMAX, and to be able to study the superiority of deep-learning techniques.
- Local visualization entails real-time and predicted health parameters, and is displayed on a local dashboard, a lightweight web-based platform, the final visualization directly on the edge device.

Within the framework of cloud integration, sensor information and model's output are uploaded by the Raspberry to Amazon Web Services (AWS) Simple Storage Service (S3). Based on this storage layer, the AWS Lambda functions act, process the data uploaded on it and trigger event-based actions. In case of an abnormal trend or a threshold being breached, Lambda will then call on AWS Simple Notification Service (SNS) to send a notification to the registered caregivers or healthcare providers through emails or SMS.

Moreover, the information contained within the cloud is integrated into dashboards of visual displays that can be accessed via the use of safe web or mobile interfaces. With the help of such dashboards, remote healthcare professionals can analyze the current data trend and past data trends, compare the predictive forecasts obtained using various models, such as SARIMAX and LSTM, and subsequently make an informed decision based on patient management.

This hybrid system architecture brings together the advantages of edge computing (low-latency local forecasting and visualization) and cloud computing (scalability, remote access, automated alerts).

The applied workflow demonstrates:

- Controlled simulation of patient data without the need for real patients.
- Comparative analysis of forecasting models using the same datasets, measured with MAE and RMSE.
- Cloud-based automation for reliable alert dissemination.
- Local resilience through web-based visualization at the edge.

By validating the system with synthetic data, this work ensures that the architecture and forecasting pipeline are robust and ready for future deployment with real-world CGM and heart rate sensors.

3.5 System Architecture

3.5.1 Structural Layout of the Proposed System

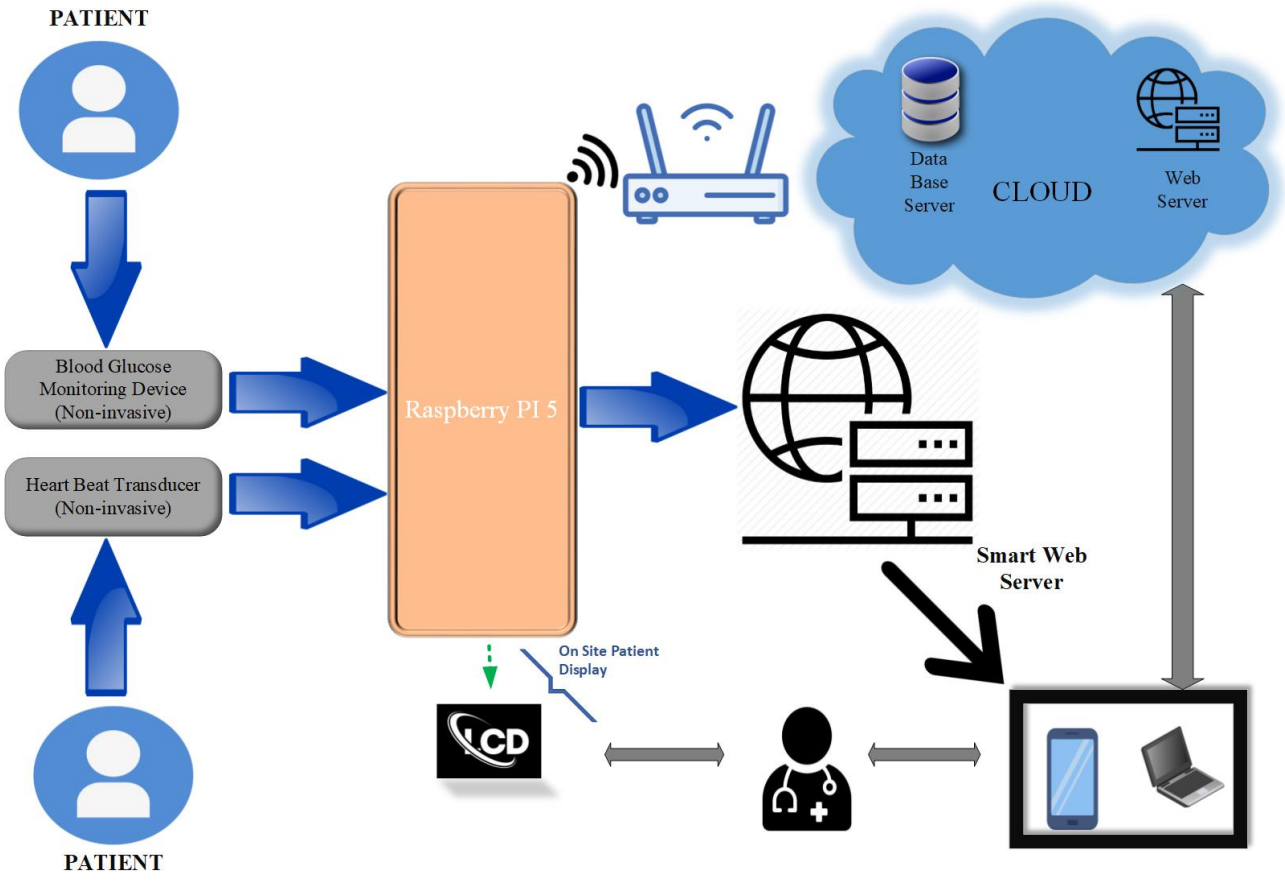


Figure 3.1: Patient Health Monitoring System Architecture With Edge-Cloud Connectivity

The figure 3.1 illustrates the overall design of the proposed hybrid Remote Patient Monitoring (RPM) system, where the Raspberry Pi 5 acts as the central processing unit. In the proposed architecture, blood glucose and heart rate values are intended to be captured through non-invasive sensors connected to the patient. However, in the current working scenario, these values are generated synthetically using Python scripts in Visual Studio Code. The simulated data then goes into the Raspberry Pi 5 where it is further processed, forecasted, and visualized.

The Raspberry Pi 5 is also set or configured at the local level as a Modbus TCP/IP server device and hence provides smooth communication of physiological parameters with other applications and

devices in the network. Real time display of the processed values in a local web controller. At this end, a graphical visualization tool (RC Studio) has been used and is a proprietary tool in order to create a user-friendly and intuitive interface. This enables both clinicians or caregivers on the ground to access patient information locally, as well as trends and forecasting outcomes of baselines without having to access any external cloud services.

At the same time, the Raspberry Pi 5 transfers the identical data to the cloud level to Amazon Web Services regularly as per assigned frequency. The information is initially uploaded to cloud storage where they are stored and sorted to be analyzed later. AWS Lambda uses them to invoke AWS Lambda, which does anomaly detection and threshold analysis on the received values. In case of irregularities, the system will activate AWS Simple Notification service (SNS), and this will send instant alerts to the caregivers or medical staff either via email or SMS. At the same time, a cloud-based web server will also be able to offer dashboards that can be viewed on mobile phones, laptops, and other connected devices, thereby being able to control the patient remotely, and in any location imaginable.

This two-layer architecture, which incorporates local visualization of Modbus device parameters on web controller dashboard with an IP and TCP interface and cloud storage as well with automation, and alerting, making the system more powerful and flexible. The Raspberry is able to ensure that local entities can see data and forecasts even in cases where the cloud connection is unstable. Simultaneously, cloud connection leads to an increase in accessibility, which makes clinicians track and react to the status of patients in a real-time environment, no matter where they are.

3.5.2 Edge Layer Forecasting and Local Processing

The proposed hybrid Remote Patient Monitoring (RPM) system includes the edge layer, having its vital role in the real-time processing and initial processing of patient health parameters. This layer is comprised of a microcontroller the Raspberry Pi 5 as the edge computing device connected directly with patient sensors.

The main aim of edge layer is to reduce the dependence on their ongoing connection to a cloud by performing critical computations on the edge. This does not only accelerate the process of anomaly identification but also enhances the resilience of the system, especially when deployed in a system with a low or volatile internet connection.

3.5.2.1 Sensor Integration and Data Acquisition

The main sensing components in the planned hybrid RPM system include a Continuous Glucose Monitoring (CGM) unit to monitor the level of blood glucose and a high-precision heart-rate sensor that can work with the Raspberry Pi 5. These types of sensors will be directly connected to the microcontroller using standard methods of communication like GPIO pins, I2C or SPI depending on the hardware setup. The CGM offers nonstop and automatic glucose monitoring, and the heart-rate sensor captures pulse data in real-time hence forming the basis of short-term prediction and alerts creation.

Physical sensors were not used in the current implementation stage due to operational limitations of getting genuine data on patients. Rather, blood glucose and heart rate synthetic datasets were created with Python scripts in Visual Studio Code. All these datasets were carefully designed to reflect actual trend in the real world including normative physiological scales, which include anomalous changes that could raise an alarm. This type of methodology provided a controlled setup to test the forecasting models and consider the full functionality of the system end-to-end.

After the process of generation, the synthetic data was then preprocessed on local microcontroller. Data normalization, setting timestamps, and sorting into CSV files to be stored in an organized way are some of the procedures that were included in the preprocessing pipeline. Such files were then used as inputs to forecasting models (SARIMAX and LSTM), which made it possible to predict patient health indicators accurately (in the short-term).

This two-designed method, which includes integration through sensors to be deployed in prospect and simulation through synthetic data to test it in present, will guarantee that the system can be effectively upscale. To deploy CGM sensors in a live setting, it will be quite natural to have the Raspberry Pi 5 to receive, pre-process, and exchange the data to both local visualization and cloud alerting.

3.5.2.2 SARIMAX-Based Forecasting

Once the data is collected, the Raspberry Pi runs the SARIMAX (Seasonal Autoregressive Integrated Moving Average with exogenous factors) algorithm to forecast short-term health trends. The SARIMAX model allows for:

- Handling of time series data with trends and seasonality
- Incorporation of external variables (e.g., meal times, medication, etc.)

- Short-horizon prediction of vitals like glucose level for proactive alerting

The algorithm operates on a rolling window of recent data to produce updated predictions in near real-time.

3.5.2.3 LSTM-Based Forecasting

Besides SARIMAX, there are Long Short-Term Memory (LSTM) networks that are used to predict the patient vital parameters, including blood sugar and heart rate. LSTM is a model of recurrent neural network (RNN), which is especially useful at capturing long-term correlations and nonlinear patterns in a sequence of health-related data, and therefore is suitable to patient-specific applications where trends are not necessarily linear or seasonal.

The LSTM model is also trained on synthetic datasets, which were made in Python and organized into CSV files. Through training, the model is trained to understand short-term variations and long-term time variation therefore it is able to internalize complex time variations that are contained in the data.

After being deployed on the Raspberry Pi the LSTM model will run inference real-time on it, giving multi-step predictions of the vital signs. The predictions are then used to predict the existence of potential risks such as health hazards before their occurrence, thus making it easier to interfere proactively into the clinical context.

The main peculiarities of the LSTM-based forecasting of this system are:

- Capacity to learn complicated, irregular and noisy data without having rigid stationarity.
- Better metrics, i.e. larger or longer-term datasets, than SARIMAX.
- Live execution of the program on the Raspberry Pi, which confirmed that it was capable of processing event in a resource constrained system.
- Comparative analysis with SARIMAX based on such accuracy measures as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

By combining SARIMAX and LSTM within the hybrid architecture, the system leverages the interpretability and simplicity of SARIMAX along with the high accuracy and adaptability of LSTM, enabling a robust forecasting framework suitable for both short-term and long-term patient health monitoring.

3.5.2.4 Threshold Evaluation and Local Triggering

The value forecasts are checked against clinically established criteria of each measurement. When any of the expected values surpass the set safety limits, a local trigger is set off. This trigger is then sent to the cloud platform to be analyzed and alerted upon of advanced features.

The Raspberry Pi 5 has the provision to locally record the aberration and send primitive notifications over an alternative channel e.g., broadcast on the local network or maintain a log record that can be later updated when offline to the cloud.

3.5.2.5 AWS-Based Logic and Alerting On Cloud Layer

The proposed hybrid Remote Patient Monitoring (RPM) architecture associates the cloud as the central focal point in making decisions, automation, and delivery of real-time notifications and dashboard to care providers. Application on Amazon Web Services (AWS) platform takes advantage of server less services, which are scalable, which increases the reliability and flexibility of the system.

3.5.2.6 AWS Lambda for Logic Processing

The given hybrid Remote Patient Monitoring (RPM) architecture places the cloud at the center of the decision making and automation process, as well as providing real time notifications and dashboards to the care providers. Implementation on the Amazon Web Services (AWS) makes use of the server less services which is automatically scalable thus enhancing the stability and flexibility of the system.

The Lambda function:

- Analyzes the incoming data against configured medical thresholds
- Determines the severity level (e.g., warning vs. emergency)
- Triggers the appropriate response, such as sending an alert or logging for future review

Lambda's server less nature allows the system to scale dynamically without requiring dedicated servers, ensuring high availability and low maintenance.

3.5.2.7 AWS Simple Notification Service (SNS)

After processing, real-time notifications are sent by being transmitted with Amazon Web Services Simple Notification Service (AWS SNS). SNS allows several channels of communication, which include

- Electronic mail messages addressed to care givers or health workers.
- Short Message service (SMS) messages or mobile push messages, the use of which is expected to be realized in future improvements.

3.5.2.8 AWS Storage Service (S3)

The system uploads the data on cloud storage using Amazon Simple storage service (S3) to make sure that important information is not just safe only but accessible to concerned medical staff when they need it. When processing or forecasting tasks on the Raspberry Pi are finished, the data generated (e.g., daily logs, synthetic datasets generated by the Raspberry Pi to verify the results, the output of the predictive model that was used, such as SARIMAX or LSTM) is sent to a separate S3 bucket.

The utilization of S3 storage can provide following benefits to enhance the reliability of the data

- Investigators and medical professionals can verify themselves at different places to access stored information.
- The service supports large amounts of data hence nullifying the fears of storage capacity exhaustion. Data integrity will be maintained even when there are local hardware failures.
- Previously archived files can be reused in future analyses, comparative examination or to augment a predictive model.

S3 is therefore a storage facility through which all computation outcomes are gathered at one locus. Basically, it is an all-round backup solution that works adjunct with the local server. This architecture can enable a retrospective analysis of the past records, their correspondence to the current findings and longitudinal tracking of health indicators. The scalability and strength of the S3 removes possible risks of storage overload or failure, which presupposes the existence of the safe and reliable tool of continuous supervision of the structure and availability of the dataset.

3.5.3 Data Communication & Processing Flow

The intricate communication process in the proposed hybrid RPM system is designed to keep it flexible. Even though in the present scenario patient data is synthesized without the physical presence of sensors, the system has been designed where physical sensors (e.g. CGM sensor or heart rate sensor) may be added at any later stage. When connected, data handling on the Raspberry Pi 5 starts immediately and thus it does not require massive reconfiguration. The main strength of the system lies in its data transfer, processing, and data storage mechanisms, including all local and cloud-based options.

3.5.3.1 Data Acquisition at the Edge

- In the present scenario, datasets for glucose and heart rate are generated synthetically using Python scripts in Visual Studio Code.
- The Raspberry Pi is configured as the edge controller and works as a Modbus TCP/IP server device. This design makes it easy to integrate with the local web server, which supports the Modbus protocol.
- As a result, whether the data comes from synthetic generation or from physical sensors in the future, the Pi ensures proper acquisition, preprocessing, and timestamping.

3.5.3.2 Forecasting and Local Decision Support

- Raspberry Pi performs its forecast with either SARIMAX or LSTM models the type of model depends on the size of the data set where SARIMAX is the best model with short sequences of data whereas LSTM is a better model with long sets of data.
- Predictive results are always updated, thus giving near real-time information of glycemic indexes and cardiac rhythms changes.
- Local web interface, visualizes these results in animated dashboards, which is affordable to provide a clear image of patient-specific information.
- Moreover, local triggers mechanisms where audible or visual alarms are activated are also provided when exceeding some defined limits, however such facilities are not operational at the present implementation.

3.5.3.3 Cloud Processing

- Alongside local processing, the Raspberry Pi sends data and forecast results to AWS S3 for cloud storage.
- In case alerts are raised, AWS Lambda processes the incoming messages to determine urgency and prepares them for notification.
- This ensures that even if the local system is not available, the cloud can still act as a backup decision-making and alerting layer.

3.5.3.4 Notification and Storage

- AWS SNS handles the delivery of notifications, currently configured as email alerts to caregivers or healthcare providers.
- The AWS S3 service acts as a parallel storage system. However, the primary storage location is the web controller while at the same time S3 ensures that all data and results are safely backed up in the cloud.
- This dual storage arrangement enables the system that historical data can be monitored at any time reducing the risk data loss considerably.

3.5.3.5 Adaptive Feedback Mechanism

- In future enhancements scenario the cloud system could provide feedback to the edge layer based microcontroller. For example, AWS Lambda could send new threshold values, sampling frequency rate, or suggesting the particular forecasting model on the basis of data profile.
- This feedback loop enables the system to be more efficient and intelligent over time, in long& short term patient data scenarios.

3.6 Tools and Technologies

A hybrid Remote Patient Monitoring (RPM) system requires a complete setup hardware as well as software that is integrated with the cloud segment. All the constituent elements play a critical role in ensuring sound data capture, processing, analysis and visualization of patient information. In this part, the chosen technologies are revealed and their expected roles, inherent abilities and reasons of their implementation in this study are explained.

The architecture design philosophy will envisage to come up with a free and scalable architecture capable of functioning independently at the edge and also utilizing cloud services to derive advanced analytics, alerting systems, and ultimately connecting clinical dashboards. Currently, it is focused on edge-based forecasting models running on a Raspberry 5 with the help of local and cloud visualization systems to support them. There has also been the provision of future integration of biomedical sensors to avoid the need to overhaul the whole hardware.

3.6.1 Hardware Components

The hardware infrastructure serves as the foundation of the proposed RPM system. It is centered around the Raspberry Pi 5 as the main computing and forecasting device, complemented by a local web server (MPW-C/RC Studio) for visualization. While sensors are not physically deployed in the current stage, the system has the capacity to connect with biomedical devices (e.g., glucose monitors, temperature sensors, heart rate/SpO_2 sensors) at any time in future practical applications.

3.6.1.1 Edge Device Raspberry PI 5

The main processor of the system is the Raspberry Pi 5. It is endowed with data acquisition, the implementation of forecasting models, and with a task to be a Modbus server in the context of local communication. Its computing power was sufficient to effectively do SARIMAX as well as LSTM forecasting models even when used on massive data. The validation showed very satisfactory performance with scripts being firstly trained on datasets in a desktop computer environment and then run on the Pi.

The low cost, small size, and energy-efficient design make the Pi extremely appropriate in infrastructural patient care processes especially in practical patient monitoring settings where portability and sustainability are the most critical factors. Besides, its flexible connectivity features

such as Ethernet, Wi-Fi, sensors sensor interfaces, and displays HDMI interfaces, make it easily integrate with biomedical equipment and display systems.

Prominent Features

- Quad-core ARM Cortex-A76 processor @ 2.4 GHz
- 4 GB LPDDR4X RAM for efficient model execution
- Built-in Wi-Fi 802.11ac, Bluetooth 5.0, and Gigabit Ethernet for fast connectivity
- 40-pin GPIO header for direct connectivity of field devices or biomedical sensors
- USB 3.0 and PCIe 2.0 which allows connecting in high-speed data transfer and expansion
- Dual 4K HDMI outputs, and allowing real-time visualization to be optional.
- Low power usage, which makes the device appropriate in the constant healthcare processes.

3.6.1.2 Smart Web-Based Controller

Visualization is an important element of any patient monitoring system, since raw data and predictions should be presented graphically and in understandable form and dashboards. To this end, a local web controller (MPW -C using RC Studio software) is used in the system.

The smart controller is an independent web server that can support dashboards, alarm display, and trend generation. The interface is accessible via a static IP address with the help of a mobile device, laptop, or desktop computer to any authorized user. Role and permission ensure that clinicians, caregivers, and patients only see the information, which is relevant to them. The system will be easy to use, role based and it is reachable through any standard web browser without having to use specialized software.

Prominent Features

- Supports multiple protocols, including BACnet, Modbus RTU/TCP, HTTP, SMTP, and SNMP for easy integration with devices and systems.
- Provides a built-in web server with HTML5 dashboards and graphics, accessible from any browser on laptops or mobiles using its IP address.
- Offers real-time control with PID loops, trend logging, schedules, and calendars for automation and monitoring.

- Allows secure multi-user access with up to 128 accounts and TLS/SSL encryption for safe communication.
- Expandable design with support for additional I/O modules, large data handling capacity, long-term memory backup, and international certifications ensuring reliable operation.

3.6.1.3 Forecasting Models SARIMAX and LSTM

The main components of the research are two forecasting models, SARIMAX and LSTM. The first tests conducted in the controlled desktop setting were, to test the accuracy of these models and performance. They were validated by attempting and deploying them on the Raspberry Pi5 to execute in real time.

- **SARIMAX** (Seasonal Auto-Regressive Integrated Moving Average with Exogenous Variables) is employed in orderly time-series forecaster. It is especially very powerful on smaller datasets whose temporal trends are obvious.

Prominent Features

- Captures seasonality and long-term trends effectively.
- Incorporates external influencing factors for more realistic forecasts.
- Provides transparent and explainable parameters.
- Works reliably with smaller and structured datasets.
- Performs efficiently without high computational demand.

- **LSTM** (Long Short-Term Memory networks) is a deep learning approach, was chosen for its ability to handle long-term dependencies and complex datasets. Although computationally heavy, the Raspberry Pi 5 successfully executed the model, proving its suitability for edge-based healthcare forecasting.

Prominent Features

- Learns long-term dependencies across time sequences.
- Handles non-linear and sudden changes in data patterns.
- Scales well with large and complex datasets.

- Improves accuracy as more data becomes available.
- Remains robust even with noisy or irregular sensor data.

Both models enable the system to predict future health trends of patients such as blood glucose levels or heart rate in a specific scenario, which triggers the alerts on the basis of abnormal conditions if forecasted.

3.6.1.4 AWS Cloud Services

While edge devices handle local forecasting while at the same time cloud services provides reliability, and remote accessibility. The system uses Amazon Web Services (AWS) for automation, notifications, and storage.

- AWS Lambda: Executes server less functions that process incoming alerts or forecast results from the edge device.
- AWS SNS (Simple Notification Service): Sends automated alerts, such as email or SMS, to healthcare providers in case of critical conditions.
- AWS S3 is responsible for storing patients data, alerts stats, and logs for further long-term analysis

These services ensure the communication reliability of critical health events while also building a secure data repository for future clinical evaluation.

3.6.1.5 Display and Visualization Devices

The system, supports visualization on any compatible display.

This includes

- Direct HDMI monitors connected to the Raspberry Pi.
- Laptops or desktop computers through local IP or AWS dashboards.
- Smartphones and tablets through web browser access.
- Direct access via Ethernet port on any PC or laptop via assigned IP address

This flexibility ensures that users, whether patients, caregivers, or clinicians, can access dashboards and alerts through devices they already own, without needing specialized hardware. In offline

scenarios, the Raspberry Pi with an HDMI display can serve as a standalone interface, while in online deployments, AWS dashboards provide remote access.

3.6.2 Software & Development Tools

The section defines the software architecture that was used to design, deploy, and administer the hybrid RPM system. It looks at the choice of programming language, program development environments, libraries used, cloud-based applications and proprietary program implementations used in designing the dashboard. All these tools are used to enable data forecasting, and automation on cloud platforms, alarm generation, and advanced visual representation to render healthcare monitoring.

3.6.2.1 Programming Language

The programming language was selected to be Python due to its ease of use and flexibility and numerous data-science and Internet-of-Things libraries that make it a very strong tools platform. Implementation of both the models the SARIMAX and LSTM forecasting while other features like data preprocessing, anomalies detection, and contacting with the cloud services is also done by python.

3.6.2.2 Development Environment

The initial development and debugging were carried out in Visual Code Studio using python to ensure the more user friendly atmosphere for programmer.

Once tested and validated, the Python scripts for SARIMAX and LSTM were deployed on the microcontroller at edge layer for real-time execution ensuring that the models could run efficiently in actual site conditions.

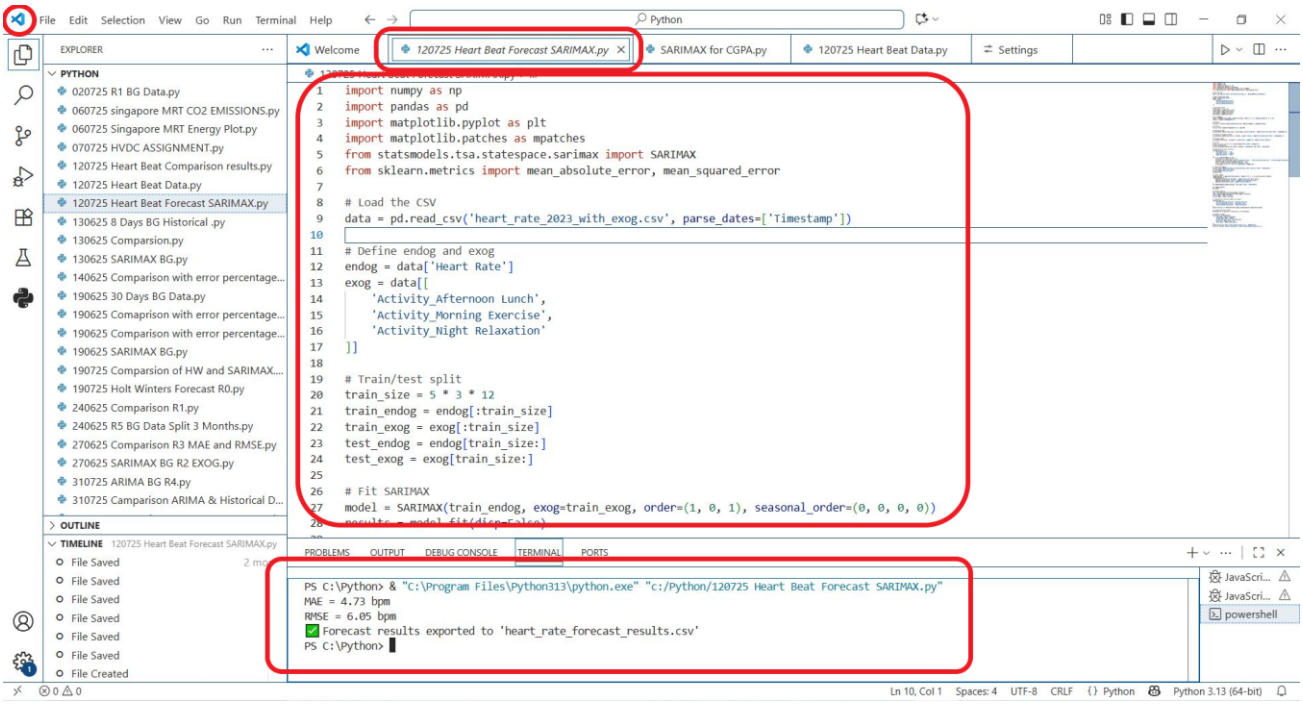


Figure 3.2: Visual Code Studio Simulator for Python

3.6.2.3 Python Libraries

Python libraries play a major role in this system because they provide ready-made tools for data handling, forecasting, and cloud integration. Instead of building everything from the base, these libraries allow efficient development, reduce complexity, and improve accuracy. They are essential for each and every function that is implemented in simulation and real world environment.

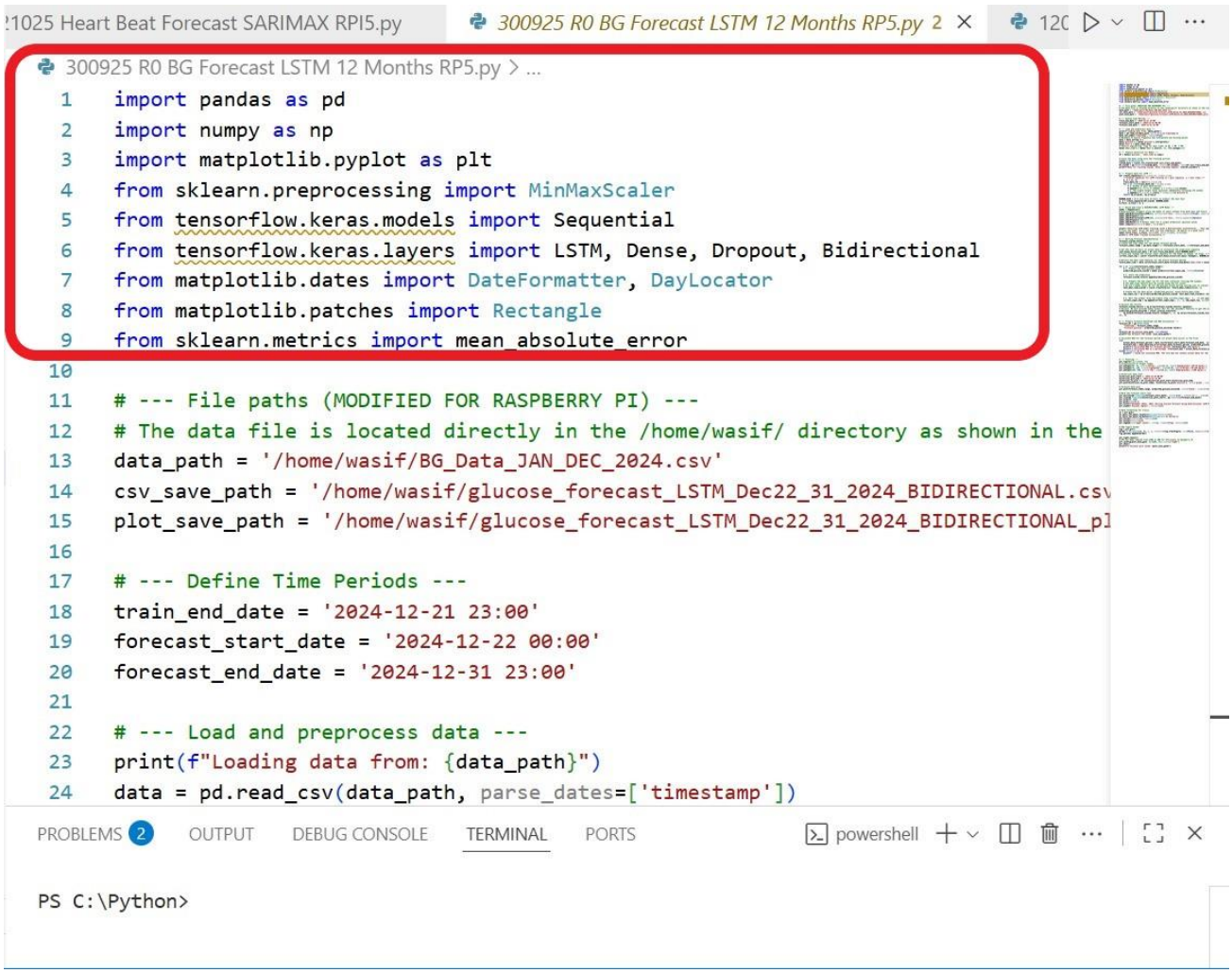


Figure 3.3: Programming Environment in Visual Studio Code Showing Required Libraries

Following libraries are very widely used in SARIMAX and LSTM

- **Stats Models**

The Stats Models is an efficient method of applying a SARIMAX forecasting model, which is capable of considering seasonality, trends, and exogenous variables in time-series data and makes it specifically convenient with patient health-related parameters. With the help of Stats Models, the system generates precise short-term predictions which allow detecting anomalous changes in vital signs.

- **TensorFlow**

A deep-learning platform called TensorFlow, being used everywhere, allows building LSTM models. It offers the building, training, and fine-tuning utilities of neural networks, which can capture long-term dependencies in the data of the patients. In this project, the use of TensorFlow supports advanced forecasting activities that rely on the existence of non-linear patterns in non-linear patterns of heart-rate and blood-glucose measurements.

- **Scikit-learn**

The functions principal to scikit learned are preprocessing and evaluation functions. It provides the working process with the utilities of dividing datasets into training and testing groups, scaling numerical values, and the measurement of the performance metrics represented as MAE and RMSE. It is also a minor contribution, but it is needed to realize the integrity and reliability of the LSTM pipeline.

- **NumPy**

NumPy is the library on which the efficient manipulation of huge numerical data is based. It provides fast mathematical calculations on array and matrices, which are essential in handling continuous inflows of health information. The result is the lightweight speedy computations on Raspberry Pi platform due to its seamlessly interoperable character with the other libraries.

- **Pandas**

The Panda app helps to convert raw sensor data to organized table structures (Data Frames). It also provides the cleaning, indexing, and resampling of the data before feeding it through SARIMAX or LSTM models, making time-series data easier to handle, visualize and analyze.

- **Boto3**

Cloud integration is enabled using the official AWS SDK in Python; Boto3. It makes it easier to invoke AWS Lambda functions, send alerts via Amazon SNS and communicate with Raspberry Pi-based local execution, and cloud-based automation to make sure communication between the local operation and the cloud-based one is as transparent as possible.

3.6.2.4 Cloud Services

Cloud services are an essential part of this hybrid RPM system because they extend the local capabilities of the Raspberry Pi to a reliable and scalable environment. While the edge device handles real-time processing, cloud integration ensures long-term storage, remote accessibility, and automated alerting. By using AWS services, the system can process events without maintaining heavy infrastructure, send timely alerts, and securely back up patient health data. This combination of edge and cloud provides both speed and reliability, making the system practical for real-world healthcare monitoring

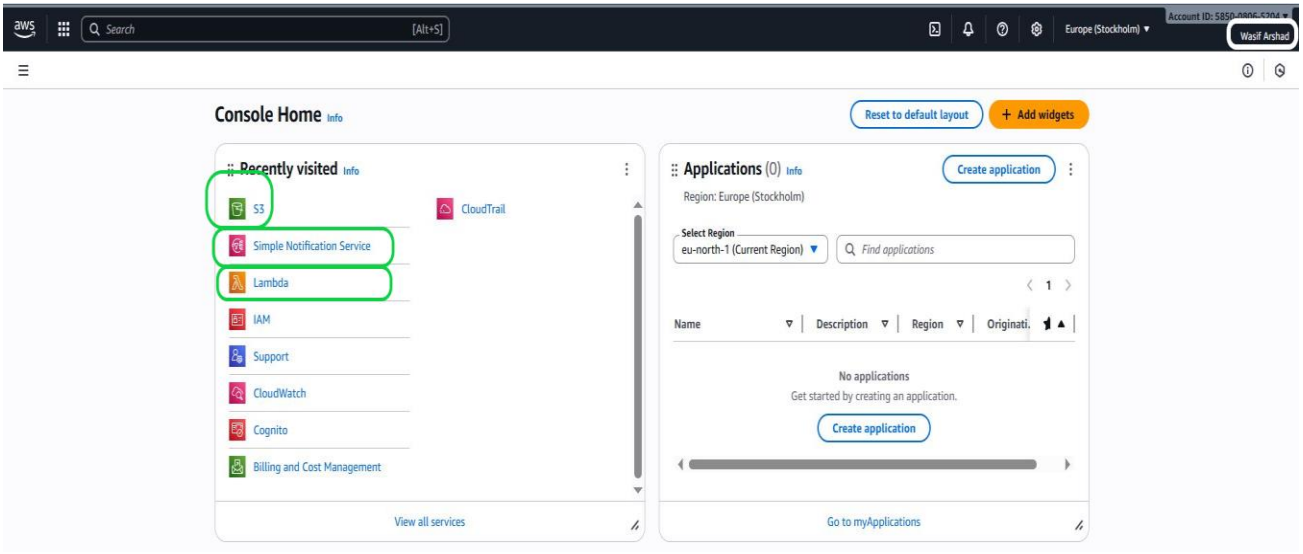


Figure 3.4: AWS Console Showing Essential Services Used in Real-World Health Monitoring

- **AWS-Lambda**

AWS Lambda is the processing based on which the system operates, as new pieces of code are executed each time new patient information is uploaded to the cloud. It is not run continuously, but rather driven by an event hence maintaining cost-efficiency. The first benefit is the fact that it can verify incoming physiological measurements, such as heart rate or glucose levels, against pre-determined values instantly. After the incident Lambda detects as anomalous readings is detected, it automatically triggers consequent actions, which may include sending alerts or initiating specific analysis. Additionally, its ability to scale easily, not only to track a single person, but also hundreds of them, makes it easy to maintain stability in performance under diverse loads. The mechanism is

therefore especially valid in healthcare situations where latency as little as that can be critical. More importantly, the server less architecture eliminates the need to maintain servers or issues of downtime, making the system be lightweight and powerful. Altogether, Lambda can streamline the process of monitoring, improve its reliability and save costs at the same time, remaining real-time responsive.

- **AWS SNS**

AWS SNS is the layer in the system that uses messaging and notification. In the event where Lambda identifies an abnormal physiological condition, SNS will be effective in providing the information to the right people at the right time. Alerts are mostly sent through email but depending on the necessity the service may be expanded to include SMS or be incorporated into infrastructures used in hospital management. This flexibility will ensure that patients and clinicians are in touch and communication. The second strength is its high reliability; SNS is designed with high availability that will not cause delays and loss of patient measures. The service also allows simultaneous delivery of messages to various recipients which is beneficial when it comes to informing the patient as well as the medical team at a given time. The ease of configuration, as well as high reliability, makes SNS suitable in the healthcare monitoring where the warmest communicative speed is crucial. By closing the distance between detection and action, SNS ensures the prevention of any abnormal trend during a timely response, which improves the safety of the patients and strengthens trust in the system in general.

- **AWS S3 (Simple Storage Service)**

Simple storage solution to this system is AWS S3 (Simple Storage Service) with long-term and secure storage. Since the Raspberry Pi has a limited capacity and is not suited to long-term data processing, in the case of the data processing, S3 ensures that information about patients is saved in the cloud. Sensor readings, CSV logs and health data are regularly synchronized to ensure there is an insurance against device failure as well as unintentional loss of data. As a result, despite power outages or equipment malfunctions within the local system being experienced, patient information is safe and available. Another significant benefit is scalability S3 can handle a large amount of structured and non-structured data without filling out stored memory. This characteristic makes it both an appropriate tool to use during regular monitoring and a possible tool to create a broad longitudinal resource of patient health. In the long term, these historical data points are invaluable as far as more sophisticated analytics, trend detection, predictive modeling or even machine-learning uses are concerned. So by

the help of centralized storage, the sharing is enabled across various systems related to healthcare providers. So S3 provides a reliable back-up system and support of future developments, increasing system reliability considerably.

3.6.2.5 RC Studio (Dashboard Development Tool)

The system includes an advanced graphical framework that is designed to create user interfaces and visualizations. RC-Graphics Set editor, enables users to make dynamic as well as static graphics which enables the advanced abilities of animation in equipment, live dashboard, charts and gauges.



Figure 3.5: RC Studio Graphical User Interface Developer Software

3.7 Forecasting Framework

Forecasting is a necessary element of the hybrid remote patient monitoring (RPM) system that transforms patient sensor data into inferences. The forecasting framework not only use the measurements of the current or past but forecasts future trends in crucial parameters of blood glucose. This foresight ability helps to detect aberrant fluctuations early on, represents the foundation of making quality clinical decisions in time, and reduces the risk of medical emergencies. By combining both the SARIMAX and long short-term memory LSTM models, the system combines both the statistical rigor with the deep learning flexibility thus giving reliable, context-aware predictions which can be considered and applied in actual healthcare monitoring.

3.7.1 Introduction to Forecasting in the Hybrid RPM System

In the proposed hybrid remote patient monitoring (RPM) system, forecasting takes center stage, as it becomes necessary to close what is otherwise a disparity between acquisition of raw sensor data on one hand, and actionable clinical information on the other hand. Whereas real-time surveillance makes it easier to realize the occurrence of immediate physiological deviations, forecasting provides the system an opportunity to project any risk that may arise before being manifested. The system forecasts vital physiological values such as blood sugar levels and heart rate to provide a prospective view of the vital parameters to both depict the clinical staff and the patient with an enhanced health care plan, taking the responsibility of proactive health management. Such predictive ability reduces the chances of sudden emergencies because it allows implementing therapeutic regimens, pharmacological interventions, or behavioral changes in time.

The hybrid architecture also provides an additional benefit by extending the forecasting capability, since local edge computers such as Raspberry Pi computers can perform initial forecasting analysis in near real time, hence ensuring instantaneous responsiveness despite the lack of real time connectivity to the cloud. At the same time, cloud based services enable model optimization over a long period, historical studies, and optimization of the operational performance. Together, both levels make sure that forecasting ceases to be a side functionality, and is part of the larger decision-making system.

Finally, forecasting module transforms system into a much more proactive responder to abnormal physiological readings rather than that of a mostly reactive nature, which is the main goal of the system to improve the patient outcomes in the lack of chronic disease care.

3.7.2 SARIMAX Forecasting Model

Circadian rhythms are also described in both blood glucose and heart rate. Blood glucose levels will increase after meals and decrease in fasting periods, and heart rate varies in accordance with circadian, affect, and physical activity. These oscillations are normally periodic and repetitive and require forecasting models that can capture such seasonality. SARIMAX is since seasonally differentiable as well as parameter tunable.

3.7.2.1 Introduction to ARIMA & SARIMAX

ARIMA or Autoregressive Integrated Moving Average is one of the well-known models of time-series prediction used to forecast next time-values by analyzing the values in the past. Indeed, although it is quite effective due to stationary and periodic series, ARIMA does not support seasonal cycling and is not capable of incorporating external factors, that is exogenous variables.

SARIMAX or Seasonal ARIMA with exogenous variables has enhanced the capabilities of ARIMA with regards to seasonality and the use of extra explanatory variables. The above improvements make SARIMAX especially applicable to the prediction of physiological signals, like blood glucose and heart rate that are influenced by periodic processes and external stimuli like a meal, drugs, stress, or exercise.

3.7.2.2 Seasonal Patterns in Blood Glucose and Heart Rate

The glucose level in the blood and the heart rate show clear circadian rhythms. There is also a typical increase and decrease in cardiac frequency under the influence of circadian drives, affective states, and exercise, but the postprandial glycaemia tends to be high and returns to normal during the fasting. Such oscillatory processes have often been repetitive and periodic and therefore the need to have a forecasting framework that could capture such seasonality. SARIMAX provides the property of seasonal differentiations and parameter optimization and thus, the correct estimation of these empirically recorded patterns.

3.7.2.3 Limitations of ARIMA in Medical Forecasting

Even though the ARIMA model is effective in the modelling of simple univariate time series, it is not effective in the clinical environment, where the determinants affecting the observed vitals are numerous. Its natural shortcoming is inability to absorb exogenous information, therefore it is not

suitable for the complex body signals orchestrated by endogenous factors which are very rare. On the other hand, SARIMAX model considers such external covariates thus providing a more practical and detailed forecasting model of real-time patient monitoring systems.

3.7.2.4 Accurate Forecasting for Clinical Settings

High predictive precision plays a crucial role in clinical practice, where it is necessary to undertake timely alerts and interventions. Results of perturbations in blood glucose or heart rate can be precisely predicted to provide the system with information that can be used before the individual has to face dangers of hypoglycemia among other abnormalities. The SARIMAX model also provides the ability to add seasonal variations and the relevant exogenous modification, including insulin administration, physical exercises and psychological stress which will increase the dependability and practical uses of the prognostications.

3.7.2.5 Why SARIMAX is Chosen

The forecasting engine of the hybrid RPM system was selected to be SARIMAX due to the fact that it is more reliable and flexible compared to simpler statistical models. Its major strength is its ability to express recurrences in medical data, e.g. daily cycles in blood glucose, and at the same time the presence of exogenous factors, e.g., meal timings, insulin dosage and physical activity. Therefore, it will be particularly applicable in-patient monitoring situations where thousands of variables appertain to the movement of vital signs.

Another obvious reason is that SARIMAX already knows how to make short-term prediction when the accuracy is much needed to identify the sudden changes or negative preconditions. The model provides very comparable predictions to those of observed data and therefore reduces cases of unnecessary alerts and ensures timely warning in situations of real aberrant.

Lastly, the operational approach of SARIMAX is debated to be high-performance in edges, like the Raspberry Pi 5, therefore it makes it an operationally viable solution in real-time monitoring. This has made it a reliable component of the decision-making structure of the hybrid RPM system because of its excellent accuracy, interpretability, and consumption of computational resources.

3.7.2.6 Process Flow of SARIMAX Forecasting Model

A systematic workflow has been taken in the application of the SARIMAX model in order to ensure the ability to predict accurately the crucial physiological parameters such as glycemic levels and cardiac rhythm. The process will start with obtaining raw time-series data as collected by wearable sensors or clinical recordings and will be divided into several steps, like data preprocessing, model training, optimization of the hyper parameters, forecasting, and, finally, performance assessment as well as the detection of anomalies. The different stages are central in converting raw physiological signals into actionable information that may be used to act and adopt timely clinical actions. The figure below visually shows the chronological steps of the SARIMAX-based forecasting, therefore, the main parts and connections between them in the general analysis pipeline.

The anticipatory power of the SARIMAX structure allows making the shift to proactive patient care instead of a reactionary one. Internet of things Remote monitoring systems based on SARIMAX forecasting tools can be easily implemented with notifications to provide timely warning and support prompt therapeutic response. Proper prognostication promotes customization of therapeutic therapy, enhancement of pharmacokinetic dosage, and averting of acute morbidity amongst patients with chronic illnesses like diabetes mellitus or heart diseases.

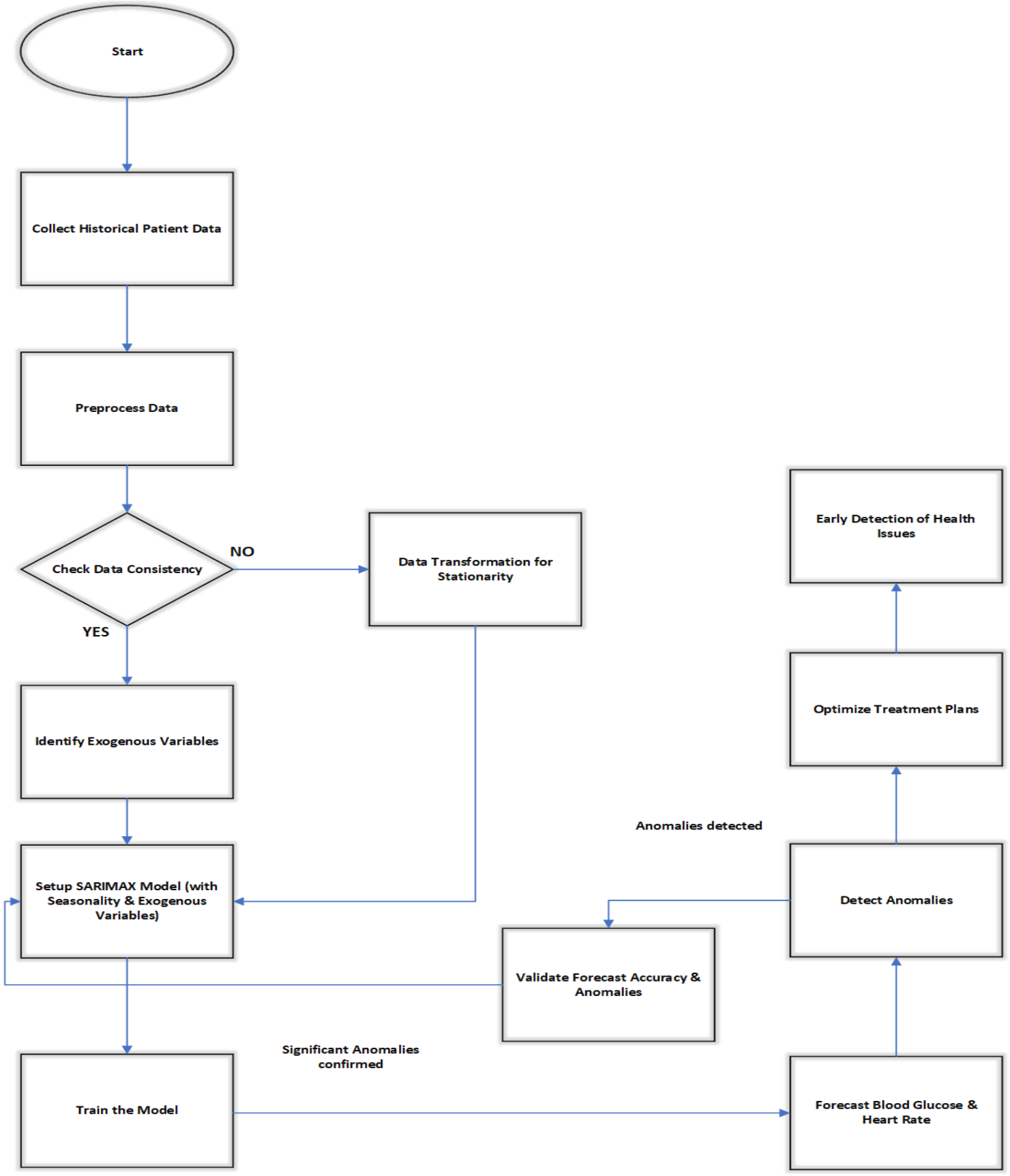


Figure 3.6: SARIMAX Model Architecture

SARIMAX (Seasonal Auto-Regressive Integrated Moving Average with exogenous variables) model is the extension of the standard ARIMA approach, which introduces the element of the seasonality and exogenous predictors. That is why it is particularly relevant to the analysis of time-related data in remote patient monitoring contexts where repetitive trends could be observed throughout a specific time period (e.g., daily, weekly) and other auxiliary tools (e.g., pharmacologic prescriptions or activity) have a control effect on the trajectories of vital signs. Below is a systematic description of the SARIMAX, which follows the diagram above.

- **Start**:

  This marks the beginning of the process, where the goal is to leverage SARIMAX for forecasting and anomaly detection in patient health data.

- **Collect Historical Patient Data**:

  Historical data is gathered, including time-series measurements like blood glucose levels, heart rate, and other vital signs. This data forms the foundation for analysis.

- **Preprocess Data:**

  The collected data is cleaned and prepared for analysis. Steps include handling missing values, removing outliers, and normalizing the data to ensure consistency.

- **Check Data Consistency**

  - The data is checked for stationarity
  - If the data is not stationary then it is transformed
  - If the data is stationary, the process moves further to identify exogenous variables.

- **Data Transformation for Stationarity**

  Non-stationary data is transformed using techniques such as differencing to stabilize the mean and variance over time.

- **Identify Exogenous Variables**

  SARIMAX allows the inclusion of exogenous variables (external factors) that may influence the target variable (e.g., medication timings, physical activity, diet). These variables are identified and incorporated into the model.

- **Set up SARIMAX Model (with Seasonality & Exogenous Variables)**

  The SARIMAX model is configured with the following parameters:

  Non-seasonal components (p, d, q):

  p: Order of the autoregressive (AR) term.

  d: Degree of differencing to achieve stationarity.

  q: Order of the moving average (MA) term.

  Seasonal components (P, D, Q, s):

  P: Seasonal AR order.

  D: Seasonal differencing.

  Q: Seasonal MA order.

  s: Length of the seasonal cycle (e.g., 24 for daily patterns in hourly data)

  Exogenous variables: External factors are included to improve forecast accuracy

Train the Model

The SARIMAX model is trained by using the past data and comes up with patterns, trends, and seasonal impacts and taking exogenous factors into consideration. Refer to Appendix A (pseudo-code 03) for implementation details.

Predict Blood Glucose

The trained model is used to predict the values of vital signs in the future, including blood glucose and heart rate, and give a reference point of what values should be expected.

Detect Anomalies

The predictions made by the model are against actual observations. Any major deviations (residuals) are indicated as possible anomalies, which is an indication of unusual patient condition.

Report Accuracy of Forecasts and Anomalies

The anomalies identified are strongly checked and verified to verify their importance. False positives are reduced using such techniques as threshold-based or residual analysis methods.

Major Irregularities Affirmed

Confirmed anomalies are affirmed as the signs of possible problematic health, including the presence

of abnormal spikes in blood glucose or an unstable heart rate.

Precise detection of health complications:

Supervision of anomalies can assist healthcare providers to early intervene thus avoiding complications and positively impacting patient outcomes.

Optimize Treatment Plans:

The model and anomaly detection are used to customize treatment plans, modify medications or provide lifestyle changes to lead better lives.

3.7.2.7 Mathematical Model of SARIMAX

- **Representation of Seasonal and Non-Seasonal Dynamics in SARIMAX**

  The SARIMAX model, which stands for Seasonal Auto Regressive Integrated Moving Average with exogenous regressors, is an extension of the ARIMA model that incorporates both seasonality and exogenous variables (if applicable).

The model is defined as:

SARIMAX(p,d,q)(P,D,Q,s)

Where:

- **p**: Number of **autoregressive (AR)** terms (non-seasonal).
- **d**: Degree of **differencing** required to make the series stationary (non-seasonal).
- **q**: Number of **moving average (MA)** terms (non-seasonal).
- **P**: Seasonal autoregressive (AR) order.
- **D**: Seasonal differencing order.
- **Q**: Seasonal moving average (MA) order.
- **s**: The seasonal period (e.g., 24 for daily cycles or 12 for monthly cycles).

The SARIMAX (Seasonal Autoregressive Integrated Moving Average with exogenous regressors) model is a strong paradigm of results and prognostication of periodical dynamics of time series of regular and seasonal character. It holds a unique notation of the form SARIMAX (p, d, q)(P, D, Q, s) adopted to divide the phenomena of time into non-seasonal and seasonal parts. The non-seasonal parameters (p, d, q) specify non-seasonal parts thus picking up the short run dependencies, effecting the differencing to remove linear trends, and picking up contemporaneous residual or lagged way up the capabilities. The upper-case parameters (P, D, Q) on the other hand with the addition of seasonal

cycle length s model the seasonal aspect which allows identification and integration of resonant recurrent on long horizons. This bi-polar design gives the SARIMAX model the ability to effectively cope with complicated time-dependent information, particularly in cases when long-term trends and cyclical seasonality co-exist to contribute to the observations, which makes it very versatile in a wide range of research fields.

3.7.2.8 Equation and Components of the SARIMAX Forecasting Model

The general equation of the SARIMAX model for a univariate time series Yt with optional exogenous repressors Xt is expressed

$$\Phi_P(L^s)\phi_p(L)(1-L)^d(1-L^s)^D Y_t = \Theta_Q(L^s)\theta_q(L)\varepsilon_t + \beta x_t$$ **Equation -3**

Where

Y_t represents dependent time series variable

X_t represents exogenous variable

L represents lag operator

s represents seasonality period

$\phi_P(L)$ represents non seasonal AR (autoregressive) operator of order p

$\Phi_P(L^s)$ represents seasonal AR (autoregressive) operator of order P

$\theta_q(L)$ represents non seasonal MA (moving average)

$\Theta_Q(L^s)$ represents seasonal MA operator of order Q

$(1-L)^d$ represents the non seasonal differencing operator

$(1-L^s)^D$ represents the seasonal differencing operator

ε_t represents the error in term (white noise)

β represents the coefficient for the exogenous regressor

Explanation of components

- **Non-Seasonal Components**: p, d and q are the parameters that define the autoregressive, differencing and moving- average components respectively. They summarize underlying structure of trend and short-term dependency in the series.

- **Seasonal Components:** P, D, Q and s are used to depict the seasonal behavior containing the cyclic behavior (e.g. the variation by the day) and the long-term trend behavior.

- **Exogenous Variables**: These are optional, but are required in medical practices. Such variables like intake of medications or insulin are included to help to predict health indicators like insulin and blood glucose or heart rate.

SARIMAX offers a powerful inhibitory scheme in forecasting time-series data with seasonal cycles and underlying trends and is particularly applicable to time-series of a health-related issue like a blood glucose level and heart rate.

3.7.3 LSTM Forecasting Model

3.7.3.1 Introduction to Deep Learning for Time-Series

Due to its ability to autonomously extract patterns of time-series data without requiring manual feature extraction, deep learning has become one of the most powerful in time-series data management approaches. Unlike traditional statistical models which often assume that trend, or other variables, are linear, or that seasonal behavior follows some pattern that is fixed, deep learning environments have the potential to embed more complex non-linear relationships that regularly occur in real-world health records.

In the field of remote patient monitoring, deep learning can be a particularly important element since physiological results like blood glucose and heart rate are adjusted by an extent of interconnected variables. These indicators are usually irregularly varied and may be explained by recurring cycles or elementary outside stimuli only to a certain extent. Deep learning models like Long Short-Term

Memory (LSTM) in this particular case are specifically designed to learn sequential data, which makes them highly reliable for using in the field of health to predict the health condition.

Forecasting accuracy is really improved by using deep learning. Although statistical algorithms like SARIMAX can provide high accuracy in forecasts with small learning data, but the induction of deep learning in terms of LSTM enables the enhancement of forecasting accuracy with nonlinear trends and huge data sets

3.7.3.2 Suitability of LSTM for Medical Forecasting

Many medical forecasting techniques often require large or sometimes very large data sets related to patient parameters to capture the forecasting trends accurately and precisely. Traditional models like SARIMAX is quiet effective and highly reliable with small datasets and with interpretable results but in scenarios where complexity or nonlinearity of the data increases the accuracy is decreased with higher level of error. The architecture of Long Short-term Memory (LSTM) networks helps to overcome this drawback. This ability to summarize the short and long-term dependencies makes LSTM an especially effective tool in the forecasting domain in healthcare, where the variation in the physiological conditions of a patient might take quite some time [66].

Another important benefit of the LSTM is its capability to study large and multivariate data. Conducted to study with the help of Python scripts running in Visual Studio Code and deployed to a Raspberry Pi controller, it is noticed that the use of longer datasets, in turn caused a lower Mean Absolute Error (MAE) of the LSTM compared to SARIMAX. This performance increase is an indicator of the model having the capacity to reveal hidden cycles of time and nonlinear relationships in the data. Though the current implementation plan is not connected with physical sensors, the Raspberry Pi solution has facilities to be connected to other devices like continuous glucose monitoring (CGM) thus allowing extension in the long-run to become a real-time medical data predictor.

However, implementing LSTM has more calculation needs hence being a challenge to resource-limited devices. By comparison, SARIMAX is plug-and-play and useful when only not very large datasets need to be analyzed, but it significantly loses performance when the subject matter becomes more complicated. The LSTM has higher forecasting accuracy when there is a lot of data, however it comes at the cost of increased processing demand [67]. This comparison highlights the complementary

nature of the two models in the offered framework whereby SARIMAX is the one needed to address small scale and efficient forecasting goals, and LSTM is needed to handle more complex forecasting jobs with the help of bigger datasets.

3.7.3.3 Process Flow of LSTM Forecasting Model

The medical time-series data preparation, training, and evaluation process flow, as shown in Figure 3.7 gives the LSTM forecasting model a framework on how medical time-series data should be prepared, trained, and evaluations made to ensure precision in the predictions. The three stages start with preprocessing that involves cleansing and normalizing of raw data and organizing them into sequential inputs that are partitioned into training and testing sets. The LSTM represents short-long term dependencies in a cell and a gating mechanism during training as well as the utility through optimization of the parameters that are optimized using the back-propagation through time up to the minimization of forecasting errors.

The model is then tested on hidden data to determine the accuracy and generalizability of the model after final training. The resulting predictions are used to make medical decisions, particularly in long data sets when such standard models as SARIMAX face challenges when trying to capture the nonlinear relationships. This procedure makes the LSTM very efficient and highly reliable model in healthcare prediction and friendly with the use of systems having moderate processing power.

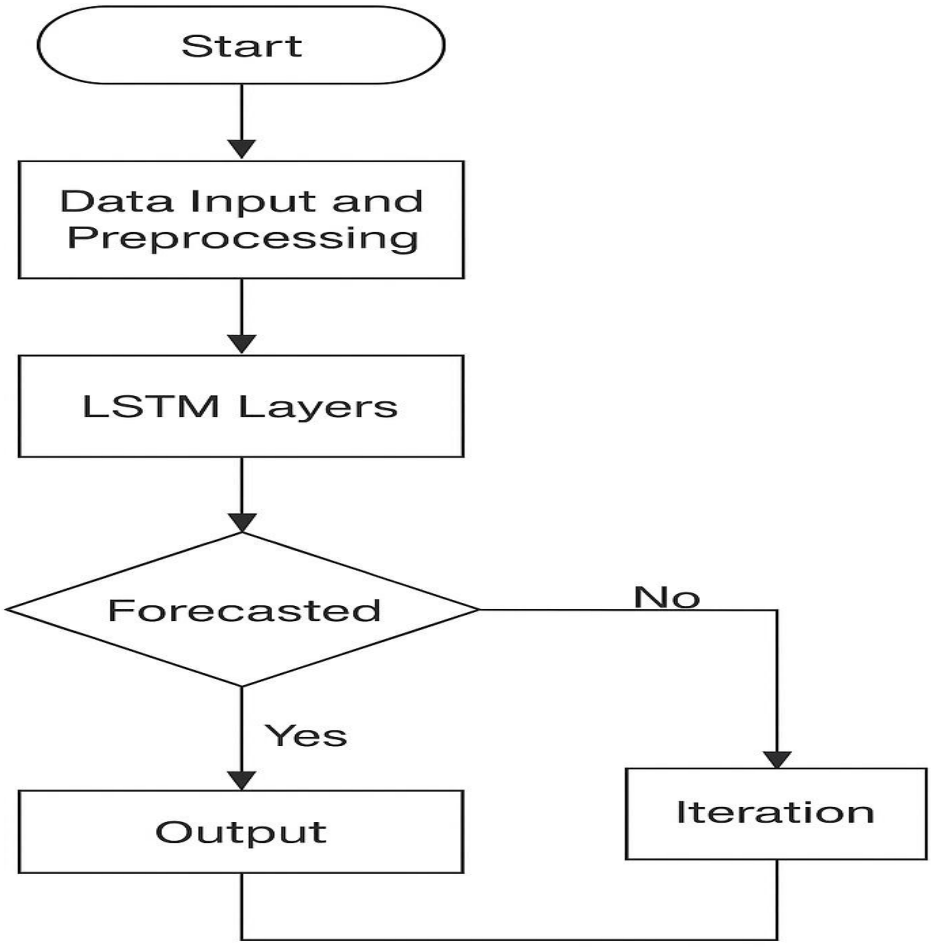


Figure 3.7: LSTM Model Flow Chart

3.7.3.4 Strengths & Challenges of LSTM

The LSTM has various advantages that make it appropriate forecasting solution to the medical data prediction. Its memory cell and gating functions facilitate the model to not only reflect the short- and long-term dependencies, which is very crucial in healthcare data sets where past events or stats may produce a strong effect on the future results. Unlike traditional models the SARIMAX and LSTM can learn even complex nonlinear relationships that exist in the data hence producing better accuracy and low forecasting errors in the case of large datasets that are continuously recorded. Its other strength is that it is versatile in multiplexing multi variables inputs hence multiple health variables can be analyzed at once using a given analytical model. Such an ability, therefore, forms a strong basis of medical forecasting related tasks where different parameters (e.g. glucose levels and heart rate) could be of interest to be tracked at the same time [68].

Even with these strengths, LSTM also has some issues. The computational cost of the training process is high, which makes it resource-intensive in comparison to other lighter methods in statistics. In microcontrollers like the Raspberry Pi 5, this may introduce longer processing time and increased energy uses and especially in dealing with large datasets. Besides, the LSTM models need large amounts of high-quality data to work, when data are scarce (or even noisy), even the SARIMAX models are likely to be more successful. Another weakness has to do with how LSTM makes its decisions. More often than not, the inner mechanics of the model can be hard to understand by a medical practitioner, who should be offered a proper explanation as to why the clinical decision was made in a particular way. These reasons reiterate the point that, despite its strength, LSTM is not a wholesome alternative to statistical models. Rather, LSTM can be integrated with SARIMAX to guarantee efficiency and strength in a wide range of data circumstances [69].

3.7.3.5 LSTM Mathematical Representation

Long Short-Duration Memory (LSTM) network refers to a deep learning architecture that will save temporal dependencies in time sequence data by its memory and gating processes. It was also trained in this paper using the same three-month simulated glucose dataset as that of SARIMAX, mainly to compare the results. Although LSTM capable of modeling complex nonlinear tendencies, it needs more data, longer training time, and computational resources, which makes it infeasible to the use of the edge of a Raspberry Pi 5 in real-time. In this case, therefore, it is used as a benchmark, and not the main model of forecasting.

The LSTM network is defined by equations that control how information flows through gates, the cell state (long-term memory), and the hidden state (short-term memory). These equations enable it to capture dependencies across time steps effectively

$$\begin{bmatrix} \boldsymbol{f_t} \\ \boldsymbol{\dot{l}_t} \\ \boldsymbol{\widehat{C}_t} \\ \boldsymbol{o_t} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma} \\ \boldsymbol{0} \\ \boldsymbol{tanh} \\ \boldsymbol{\sigma} \end{bmatrix} (\boldsymbol{w \cdot x_t + U \cdot h_{t-1} + b}) \quad \textbf{Equation-4}$$

Equation 4 computes the activations of the forget gate $\boldsymbol{f_t}$, input gate $\boldsymbol{\dot{l}_t}$, candidate cell state $\boldsymbol{\widehat{C}_t}$, and output gate $\boldsymbol{o_t}$. These gates regulate how much past information is retained, how much new information is added, and how much is exposed to the next hidden state.

$$\mathbf{C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t} \qquad \textbf{Equation-5}$$

The equation 5 updates the cell state $\mathbf{C_t}$, which serves as the LSTM's memory. The forget gate decides what fraction of past memory to keep, while the input gate and candidate state determine how much new information should be added.

$$\boldsymbol{h_t}=\boldsymbol{O_t} \odot \mathbf{\tanh(C_t)} \qquad \textbf{Equation-6}$$

Equation 6 gives the hidden state $\boldsymbol{h_t}$, which reflects the immediate memory at that moment. It works both as the present output and as input for the next step, with the output gate deciding how much of the cell's state is revealed through $\mathbf{\tanh(C_t)}$

Where

$\mathbf{x_t}$ represents input vector at time stamp t,

$\boldsymbol{h_{t-1}}$ represents hidden state from the previous step,

W & U represents weight matrix applied to inputs and hidden states,

$\boldsymbol{b}$ represents bias vector,

$\boldsymbol{\sigma}$ represents sigmoid activation (controls gates: input, forget, output),

$\boldsymbol{tanh}$ activation for candidate state and hidden state,

$\boldsymbol{\dot{l}_t f_t o_t}$ represents input, forget,

$\odot$ represents element-wise multiplication

3.7.4 Comparative Perspective of SARIMAX and LSTM

In medical time-series forecasting, no single model performs best across all conditions. Statistical models like ARIMA and SARIMAX are effective for small, structured datasets, while deep learning models such as LSTM excel with large and nonlinear data.

3.7.4.1 Key Dimensions of Comparison

- **Accuracy**

  LSTM generally provides higher accuracy on long and complex datasets, while SARIMAX is more consistent for smaller datasets.

- **Data Requirements**

  ARIMA and SARIMAX work well with limited data while LSTM requires large, high-quality datasets.

- **Model Complexity**

  ARIMA is simple and interpretable, SARIMAX is moderately complex, LSTM is highly complex and less interpretable.

- **Computation**

  SARIMAX runs efficiently on Raspberry Pi 5, LSTM is computationally heavy and better suited for GPU/cloud.

- **Suitability for Medical Forecasting**

  SARIMAX handles seasonal and external medical factors well while LSTM captures complex nonlinear dependencies in vital signs.

3.7.4.2 Tabular Comparison of ARIMA, SARIMAX, and LSTM

Table 3.1: Key Features Comparison ARIMA, SARIMAX, and LSTM for Vital Signs Forecasting

Features	ARIMA	SARIMAX	LSTM	References
Handling Seasonality	Limited	Yes, explicitly models seasonal effects	Yes, but requires custom architecture for seasonality	[71]
Inclusion of External Regressors	No	Yes, includes exogenous variables (e.g., insulin intake, meals, etc.)	Yes, via multi-input models	[72]
Model Complexity	Simple, interpretable	More complex, requires careful tuning of parameters	High complexity, needs large datasets and tuning	[72]
Ability to Handle Non-Linear Data	Limited to linear data	Moderate (handles linear relationships better, but can be combined with other models for non-linear cases)	Excellent at capturing non-linear dependencies and patterns	[73]
Suitability for Medical Time Series	Can be used, but struggles with complex relationships	Ideal for medical data with external influences and seasonal variations	Very effective for capturing complex medical data patterns, such as vital signs	[74]
Best Use Cases in Medicine	Forecasting simpler, linear data like blood glucose (without external factors)	Forecasting blood glucose levels with external factors, heart rate with seasonal variations, patient monitoring in critical care	Forecasting multivariate vital signs	[74]

3.7.4.3 Focused Practical Comparison

Table 3.2: Deployment-Oriented Comparison of SARIMAX & LSTM

Feature	SARIMAX (Statistical Model)	LSTM (Deep Learning Model)	Reference
Sustainability for edge deployment	Well-suited for real-time, low-cost deployment with limited data and resources	Less practical for real-time scenarios in constrained environments.	[14]
Performance	Stable & reliable results even with a moderate data set	Often underperforms with limited data, requires substantial historical records.	[15]
Interpretability & Simplicity	Transparent and straightforward to interpret	Less interpretable due to the complex neural network structure.	[16]
Computational Efficiency	Efficient on low-cost hardware ideal for devices like Raspberry Pi.	High computing demand usually necessitates GPUs or high-power CPUs.	[17]
Adaptation to Sudden Fluctuations	May not adapt instantly to abrupt glucose spikes.	Handles rapid and nonlinear changes better when properly trained with large datasets.	[18]

The practical differences between SARIMAX and LSTM as applied to the healthcare time-series forecasting domain are outlined by the findings provided in Table 3.2. The traditional statistical model in the form of SARIMAX has a fast computing capability, simple parameter optimization, and an easy-to-understand explanatory system, making it suitable in real-time implementations, e.g., on a Raspberry system. The fact that it incorporates capital movements through the integration of seasonal dynamics and exogenous regressors makes it tolerate fluctuations on a short time and periodic behavior precisely enough. However, it is less predictive accurate in the presence of extremely nonlinear data, or when long-horizon dynamics in patient-specific health measures exists.

On the other hand, the predictive capability of LSTM architectures is slated to be greater when dealing with complex, large-scale data since the architecture automatically learns the nonlinear tendency and long-lasting internal reliance. This benefit is specifically relevant in the case of glucose oscillation or other physiological indicators with non-regular dynamics. Majority of the trade-offs involve front-ended high computational and low interpretability as opposed to SARIMAX. This means that the LSTM settings should be adapted to a cloud-based or a GPU-accelerated system instead of small embedded systems. The comparison of these features makes it possible to assume that SARIMAX

suits localized resource-intensive applications best, and LSTM represents a more powerful tool to deliver high-accuracy prediction in centralized healthcare systems.

3.7.5 System Level Perspective

In passing the single isolated predictive models into a more integrated system, the choice of SARIMAX or LSTM is no longer limited to questions of predictive performance per se, but includes both questions of operational feasibility. The SARIMAX is particularly interesting due to its computational light weight nature, explicability, and easy deployment on edge devices like the Raspberry, so that it can be especially useful in real-time patient monitoring, at the network edges. It has the advantage of having rapid inference, which requires limited processing power and the results can easily be understood and shared with clinical personnel.

On the other hand, although LSTM networks demand higher computational resource, they provide better predictive uncertainty than their simple, linear counterparts in time dependencies that are complex and nonlinear as found in vital-sign traces. The LSTM functions best when it is on high performing servers or cloud systems in which the resource limitation factor is insignificant. Therefore, in a practical healthcare context, SARIMAX will be best utilized in in-the-field monitoring, as it requires short-term, real-time purposes, where LSTM will be utilized on a cloud platform in more long-term predictive and widespread manner.

The system also includes automated model-selection layer, in which a brief decision script chooses which forecasting methodology to apply based on dataset characteristics e.g. size and complexity. The script will give preference to SARIMAX when dealing with small/short term data to take full advantage of its speed and efficiency, and to LSTM when dealing with larger data sets with nonlinear trends. Such a dynamic fashion ensures that the system is flexible, efficient in terms of operation, and relevant in a range of medical monitoring events.

3.8 AWS-Based Automation and Alert System

This part outlines how the hybrid remote patient monitoring (RPM) system utilizes Amazon Web Services (AWS) to generate medical alerts based on real-time and predicted health-related data automatically. The use of AWS services, in particular Lambda and Simple Notification Service (SNS), allows distributing important notifications to both patients and healthcare providers with ease, safety, and at scale.

3.8.1 Trigger Logic

The logic is triggered and worked through by repeatedly comparing patient medical data with predetermined medical parameters. Predictive values produced by the SARIMAX model, which is executed on a Raspberry Pi5, are first sent to the S3 bucket along with raw values. AWS Lambda is then set to handle the repetitive retrieval of such stored data and determine whether any of the values recorded or projected exceed the specified upper or lower limits.

In the case of an abnormal trend in the historical data in the S3 bucket, i.e. with blood glucose above a critical high level or heart rate below a safe minimum rate, Lambda automatically identifies anomaly cases. When an anomaly is detected, the system initiates the alerting process, which ends with the email or SMS notification that is sent, using AWS SNS, to physicians, care givers or patients. Such an approach ensures that alerts are backed up by the current readings as well as historical data, which helps to increase reliability and define the problem that would otherwise not be detected with one-point consideration.

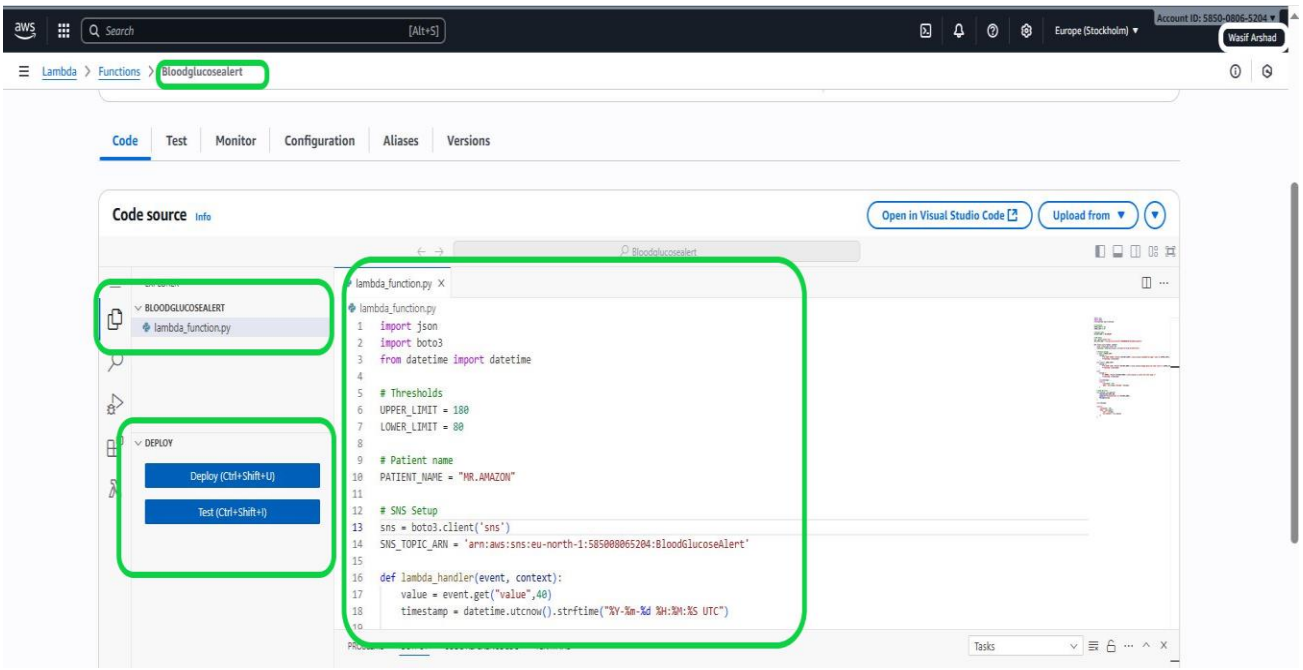


Figure 3.8: Trigger Logic Inside AWS Lambda Function

3.8.2 Lambda Setup

AWS Lambda is configured to react to the anomalies that are detected by the edge device. Once an abnormal situation has been detected, data will be sent to the cloud via either MQTT protocol or HTTP protocol. After the reception, a Lambda service is called to formulate the incoming data, reconsider the trigger logic in order to prove it and launch a due response measure.

3.8.3 SNS Configuration

After the Lambda confirmation of an abnormal reading, the system uses the AWS Simple Notification Service (SNS) in order to send immediate alerts. Such notifications are set to be sent through email or SMS to the appropriate healthcare practitioners, caregivers, or patients. SNS has the flexibility thereby enabling management of numerous recipients and notification channels. An example would be having high sugar level, requiring a physician and caregiver in the family to be notified, but sudden drop in blood sugar would generate emergency contact measures.

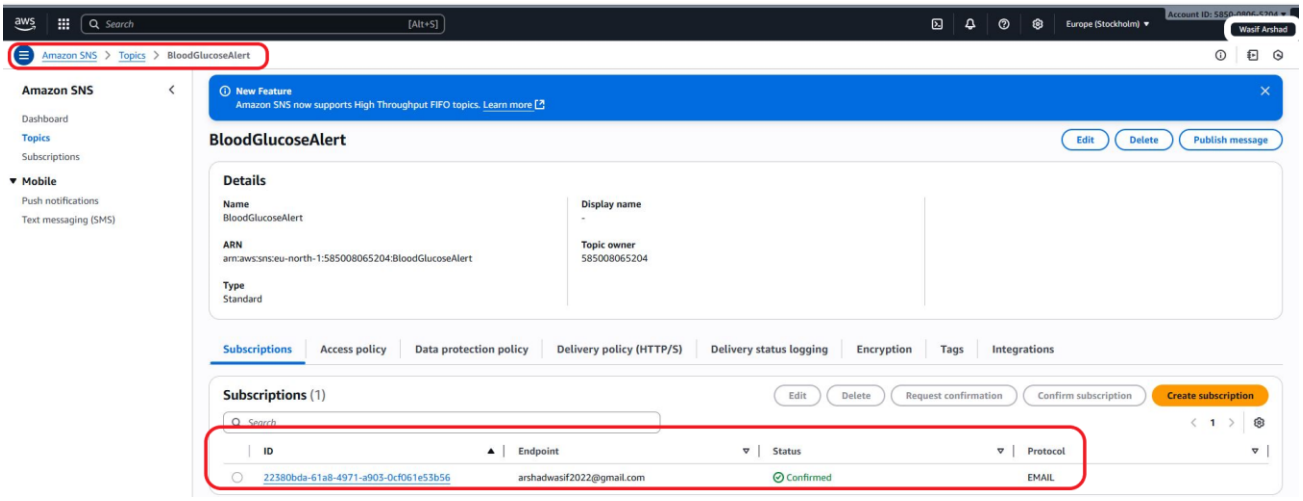


Figure 3.9: SNS Configuration for Email Alert

3.8.4 S3 Data Storage

It uses Amazon Simple storage service (S3) to store the archives of the patient health data sent by the Raspberry Pi 5 controller. All the data points, both raw sensor data and data obtained through prediction using the SARIMAX model or LSTM model, are safely consumed by an S3 bucket. This structure ensures that all data are stored in a time stamped and systematically arranged format thus ensuring that the data can be accessed easily and later processed when the need arises.

S3 has more functional advantages other than storage. The stored data may then be combined with supporting services of AWS, including Lambda to process an even greater extent and Quick Sight to visualize it, whereas the default quality of the service itself guarantees the absence of any important information. Based on this, S3 will enhance the notification system SNS offers immediate notification services, but S3 maintains a stable longitudinal document of patient information that facilitates longer-term monitoring and informs clinical decision-making.

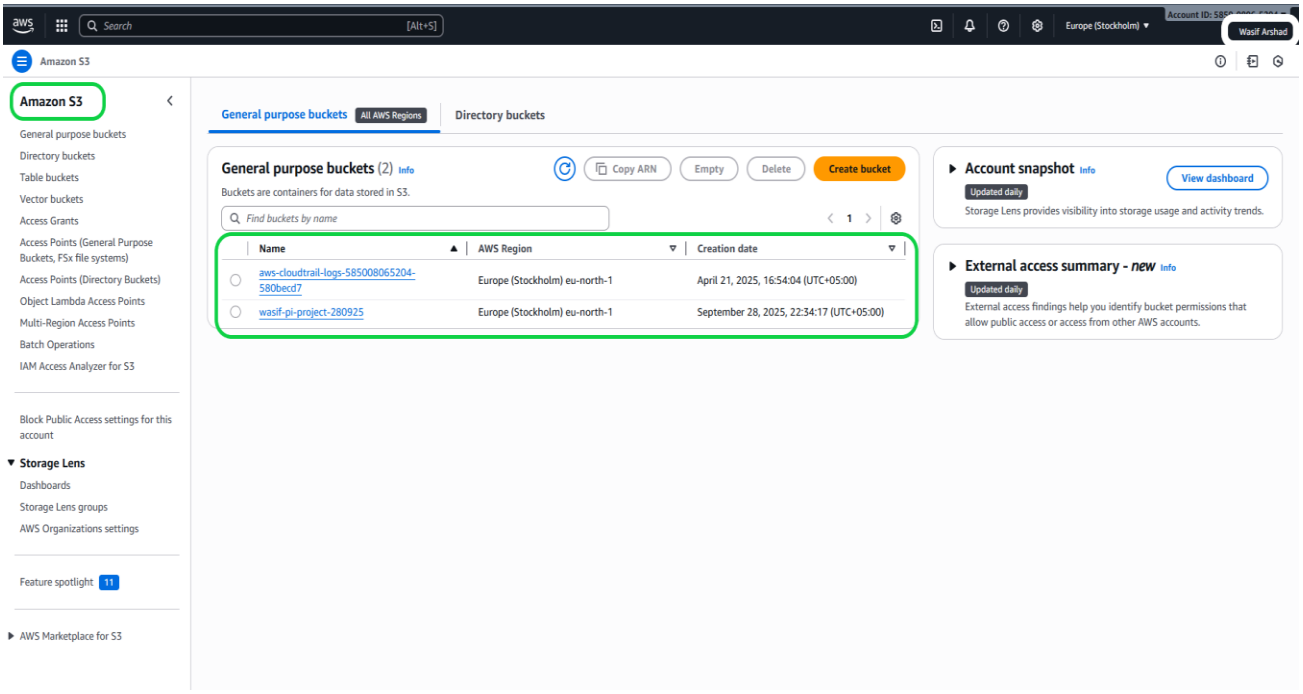


Figure 3.10: Uploaded Data Sets Inside S3 Bucket

CHAPTER 4
SIMULATIONS AND RESULTS

4.1 Introduction

The chapter contains the findings, implementation, and performance evaluation of the proposed hybrid Remote Patient Monitoring (RPM) system. Visual Studio Code (VS Code) was used as the main development environment to develop and test the system and then deployed to a Raspberry PI 5 to run in the real-time execution mode. The system edge-cloud workflow combines the data collection process with the forecasting and automatic alerts, hence, combining the local computation with cloud-based services.

First, both the forecasting models, which are SARIMAX and LSTM, were created and tested on simulated data that can be interpreted as realistic blood glucose and heart rate patterns. These models were then re-implemented on the Raspberry 5 in the environment (after completing the development phase), and real execution at the edge was possible. The above step confirmed that the system has the capability to do forecasting and simple anomaly detection without a constant connection to the cloud.

The prediction outcomes of heart rate and blood glucose were presented in terms of time-series plots that compared the observed and predicted values and thus proved that the models were accurate and capable of following the trend. Specific parameters thresholds were used to raise alerts, in the cloud layer, an alert is processed using AWS Lambda and SNS. Such functions would make sure that abnormal readings would put in real-time alerts to healthcare staff or patients.

In general, the current chapter is aimed at the validation of the RPM system simulation oriented and hardware-oriented implementation. As a result of this discussion, one can point at the accuracy of SARIMAX and LSTM predictions, the ability of the system to perform models locally in a Raspberry_Pi 5, and the efficiency of the hybrid edge-cloud architecture in maintaining the continuous and reliable health monitoring.

4.2 Simulation & Implementation Environment

This section presents the setting that was employed to develop, test, and deploy the hybrid remote patient monitoring (RPM) system. The first simulation model was meant to check the functionality of the system at a controlled setting after which the real time implementation will undergo verification. VS Code was created using Python scripts to emulate time-series data, conduct forecasting, and automatically generate alert logic by using Amazon services. This step allowed to test the SARIMAX and the LSTM models on the mock datasets that mimic the real changes in the levels of blood glucose and heart rate.

The entire forecasting framework was then set up on a Raspberry Pi 5 which was used as the edge device of the system after being tested successfully in the simulation phase. The Raspberry Pi provides real-time execution of SARIMAX and LSTM script, takes local data feeds and periodically reveals itself to AWS cloud services like S3, Lambda and SNS. This combined design provides the system to maintain its operations locally even when there is disconnection to the cloud and records the outcome to be synchronized later.

The deployment of the forecasting models on the edge device that is a raspberry pi 5 microcontroller ensures that the system efficiency and accuracy must not be degrade in the real site conditions. So it is confirmed that it performed equally good and performance doesn't degrade in real conditions where compact and robust controller is the key.

4.2.1 Mock Data Generation in Visual Studio Code

Synthetic corpora, which replicate patient physiological data in the form of blood glucose levels and heart rate, were generated through Python code that was being run as a Visual Studio Code. Such corpora are made to replicate dynamism of the real-time by including minor perturbations and anomalous events that reflect real physiological variability. This was followed by a validation of the structural integrity of the data using Matplotlib to identify the recurring patterns that could be used to perform time series prediction.

After validation, data sets are sent to an on-board microcontroller, whereby the Python scripts are used to make forecasts. The obtained predictions are presented locally and the algorithmic parameters are uploaded to cloud storage at the same time.

4.2.2 Implementation of SARIMAX & LSTM Forecasting Model

Two forecasting models SARIMAX and Long Short-Term Memory (LSTM) were implemented and compared to measure their predictive power on medical time-series technologies. SARIMAX model has captured short term seasonality and autoregressive effects and this has been used to forecast the interpretable health trends around the near future. Simultaneously, the LSTM model was patterned to learn nonlinear dependencies, as well as long-term relationships of context on the dataset, commonly found in physiological data like blood glucose and heart rate.

The two models were first trained and tested in Visual Studio Code with simulated data sets, and then run on the Raspberry Pi5 to perform real time execution. The edge device was capable of dealing with on-board computation effectively, producing temporary predictions at the same time, and holding local replicas of the results. This combination of approaches has provided an opportunity to bind the interpretability (through SARIMAX) and the adaptability (through LSTM) ingredient to the forecast property, coupled with forming better forecasting and sensitivity to changing circumstances of the input.

4.2.3 AWS Lambda, SNS, and Local Dashboard Integration

To set this up, AWS Lambda runtime functions were set to periodically execute an access request to forecast outputs in Amazon S3. Lambda function was used to compare the values of prediction with pre-established physiological thresholds. In case a critical limit of the system was surpassed (e.g. high level of blood glucose or irregular heart rate), Amazon Simple Notification Service (SNS) was triggered by the Lambda function and sent an email notification to the subscriber. This provided early notification of the possible health abnormality even where direct human observation was not present.

Besides cloud-based notice, a local visualization dashboard was created based on RC Studio on the Raspberry Pi 5. This dashboard demonstrated real time graphical results of SARIMAX and LSTM forecasts, the trends on the historical data, and the threshold-based alert. The dashboard interface was implemented on the local web controller, which allowed healthcare staff or users to view live parameters and predict trends, an interactive display was displayed without the need to stay on the internet.

The proposed hybrid architecture will be tested by this integrated solution, and data regarding the lack of scalability, responsiveness, and practical applicability of the use of this solution to real-world remote patient monitoring were tested.

4.3 System Design and Implementation in Simulation

A simulation environment was created in the Visual Studio code in order to test the functionality of a hybrid remote patient monitoring system under controlled conditions. It is designed to not only have synchronous communication between a local processing unit (Raspberry Pi 5), predictive models (SARIMA and LSTM) and cloud-based automation services (AWS Lambda, SNS, and S3 Storage Service) but also allows them to interact. All the components were configured with parameters that simulated the manner in which an operational monitoring and alerting system would behave in a manner that utilizes artificial data streams which replicate the physiological indicators of a patient which includes heart rate and blood glucose levels. This virtual environment was useful to test the system thoroughly in terms of performance, responsiveness and reliability before it was deployed in the field.

4.3.1 System Architecture

The achieved system architecture is a hybrid edge-cloud system that can be used to monitor vital signs constantly and predict vital prognostications of health. The microcontroller Raspberry Pi 5 is the core interaction point and Raspberry's computer component, as it acts as the focal point of interaction between the computation performed on the device and the services offered by cloud computing. The architecture was performed by Python scripts written in Visual Studio Code and then deployed to the Raspberry 5, so as not to introduce differences between the simulated and operational environment.

The architecture is divided into three interdependent layers which perform clear cut but interdepending functions as elaborated below.

- **Data Layer**

This layer deals with the getting and control of patient physiological information. The artificial values of blood glucose and heart rate are provided in continuous form through Python scripts and it is stored locally on Raspberry Pi. Data generated simulates real sensor behavior in real time, including built-in variation and anomalies so that a realistic simulation of physiological variations can be obtained. Once

validated, the data is then regularly uploaded to an AWS S3 bucket creating a secure repository on the cloud, which can be accessed by other modules in the system.

- **Processing Layer**

The Raspberry Pi 5 lies at the heart of architecture and thus does edge-side compression and prognostication. The execution of the both SARIMAX and LSTM models is done locally in order to predict the future health status. SARIMAX model is effective in capturing short term variations and seasonality whereas LSTM model acquires complex non-linear and long term variation using physiological data. Their results with projected values and the confidence intervals are stored and visualized onsite. This is a local calculation that reduces latency and enables resilience when dealing with connectivity interruptions and responds to potentially important health events more quickly.

- **Cloud Interaction and Visualization Layer**

The Raspberry Pi results are processed and uploaded to the AWS S3 to enable them to be remotely accessed and be integrated with the automation services of the cloud. AWS Lambda regularly checks the predicted values and compares them to set health indicators and sends alerts using AWS Simple Notification Service (SNS) in case of reminders of anomalies. Simultaneously, the Raspberry Pi has a local web dashboard designed in RC Studio that offers heart rate real-time visualizations of client blood glucose predictions. Using the local web controller, authorized users can safely log into the dashboard to display live graphs, historical trends and alert logs without necessarily maintaining constant connection to the cloud.

This layered architecture with a high degree of modularity would guarantee that there is smooth co-ordination between edge analytics, cloud-based automation, and local visualization; hence, confirming the hybrid nature of the proposed remote patient monitoring (RPM) system. The setup has been proven to be scalable, have low latency, be adaptable to real-life healthcare applications, where real-time monitoring is crucial, and real-time alert response is critical.

4.3.2 Software Components and Tools Used

Visual Studio Code or VS Code was adopted as the main integrated development environment and all simulation and implementation scripts were created and tested in it. VS Code offered an effective Python development experience, with support to use Git-based version control and immediately

deploy the Python script to the Raspberry Pi 5. Forecasting models and simulation of cloud communication were also tested using the same environment with the implementation then being transferred to the physical device. The main software part and libraries that were used to develop the hybrid RPM system are provided below.

- **Python 3**

This was the base programming language through which all the logic of data generation, forecasting and communication was implemented. The suitability of Python and integration with statistical and deep-learning packages made it perfect not only in simulation on the VS Code but also on the Raspberry Pi itself.

- **Statsmodels**

Used to employ the SARIMAX (Seasonal Autoregressive Integrated Moving Average with exogenous regressors) model. It was a library that offered trusted functions to fit, predict, and analyze short term medical time-series data, including pulse and blood sugar, to obtain variations depending on season and trends.

- **TensorFlow**

  Redeployed to construct and train the LSTM (Long Short-Term Memory) neural network, which is able to discover nonlinear and long-range physiological data dependencies. Such frameworks played a critical role of improving the accuracy and strength of such predictions as opposed to the conventional statistical models.

- **Pandas and NumPy**

The two are used to process, manipulate time-series data, address gaps, resampling time-warps, and present input data to either SARIMAX or LSTM processes. Their succinct data models had made the work with large sequences easy and fast calculations on the Raspberry Pi.

- **Matplotlib and Seaborn**

These are both applied in visualization of real-time trends, forecasted data and performance measures. There were these libraries that made it possible to generate time-series plots and comparison graphs of observed and predicted values which were further shown on the local dashboard.

- **AWS Console and Python SDK (Boto3)**

including AWS service used to automate operations in the cloud uploading forecast data in AWS S3, running Lambda functions, and monitoring SNS notifications. The Raspberry Pi executable Boto3 scripts authorized and provided secure communication between the edge layer and cloud layer.

- **Supplementary software feature: RC Studio (Local Dashboard Interface)**

This software is written to serve as the graphical interface on the Raspberry Pi 5. It has a local web dashboard that allows its authenticated users to log in and monitor the live data streams, view trends history, and read alert history. RC Studio is a real-time vocational sync with local CSV output and a visual forecast outcome with both SARIMAX and LSTM model types.

4.3.3 End-to-End Integration from Edge Forecasting to Cloud Alerts and Local Dashboard

Having successfully validated main code in Visual Studio Code, SARIMAX and LSTM forecasting scripts were implemented and run on Raspberry Pi 5, which is the central processing unit of the hybrid RPM system. The Raspberry Pi is fed with simulated sensor data of blood glucose and heart rate, locally processes each of these streams through both forecasts and, constantly compares the predicted values, to accredited clinical threshold values. This allows the detection of abnormal health conditions (on-device) before they worsen.

At any point where any forecasted or observed reading surpasses its threshold, a process of cloud communication is then automatically triggered by the system. An automation script written in Python transfers the respective health data and forecast results to Amazon Web Services (AWS) Simple Storage Service (S3) acting as an additional storage and data exchange medium and as a secure storage medium. When uploaded, AWS Lambda will be activated to calculate the historical data that is contained in the S3 and determine whether the condition is a real abnormal trend. In case, confirmed, Lambda enables Amazon Simple Notification Service (SNS) to send an alert email to healthcare personnel or registered users.

Such integration will guarantee the full closed-loop communication between the edge-level and the cloud-level, both at local forecasting on the Raspberry Pi to automated alerting in AWS. At the same time, all the projected and real values are sent to the local dashboard located in the RC Studio, it is

possible to authenticate via the local web controller there and watch real-time graphs, alerts, and system logs. Such a two-layered integration not only makes it more responsive and reliable in data handling but also allows the system to continue functionality even in the case of a network failure but keeps it in sync with the cloud services once connected again.

4.4 System Testing

The testing of the system was implemented after the successful implementation of the forecasting scripts on Raspberry Pi 5, which will be the main edge controller of the system. The SARIMAX and LSTM models were both run on the Pi to predict the heart rate and glucose concentration in the blood by using the uploaded datasets. These tests proved that the edge equipment was capable of handling the computational workload comfortably, at the same time maintaining real-time responsiveness.

The forecasts obtained were matched with preset medical limits to determine the ability of the system to identify abnormal readings. When one of the thresholds was reached, the data was uploaded automatically on an AWS S3 bucket to be securely backed up and analyzed later. At the same time, the warning system, which is integrated with the AWS Lambda and SNS, was also put to test to verify the quality of the information and the time at which the notifications are delivered to the relevant users.

The locally hosted dashboard was created using RC studio and was displayed on the web controller which showed the live forecasting outputs and the alert status. The interface allowed people to track the health patterns of patients, see their previous performance, and confirm the decision-making of the system used in a graphic interactive level.

The systematic testing phase, in general, resulted in the validity that all modules interact well, both on the edge (data generation and onboard forecasting Raspberry-Pi) and on the cloud (synchronization of data with the dashboard), thus fulfilling the strength and stability of the suggested hybrid edge-cloud system.

4.4.1 Testing Methodology & Data Flow Validation

The full hybrid setting was tested at the system testing stage, with the pieces of synthetic data and live scripts of forecasting processes running on the Raspberry 5. This testing was done to test the data flow between the edge, cloud and local visualization layers.

The controlled input data, consisting of values of heart rate and blood glucose, was presented to kick off the testing process and was therefore processed by the SARIMAX and LSTM models running on the Pi. These models came up with real-time predictions that were constantly compared with senior clinical predetermined limits. Close attention was paid to the system reacting to transgression of the threshold, i.e., data uploading to AWS 3, alerting on AWS Lambda and SNS and the visualization of the reaction on the local dashboard.

This pattern confirmed the capability of this system to provide continuous data feeds, run forecasting operations effectively on the Raspberry Pi, coordinate data with the cloud and provide real-time updates to the on-board web controller. In general, the testing confirmed the end-to-end operation and the stability of the suggested hybrid edge-cloud patient monitoring model

4.4.2 Forecasting Accuracy Validation

Both the SARIMAX and LSTM models were used to predict the heart rate and blood glucose levels to measure the accuracy of the system in its forecasting. Both models were implemented on the Raspberry Pi 5 that employs real-time simulated data and the predicted values were measured against the input series that was used to determine the performance. Both models were able to capture the overall patterns and trends albeit that there were minor differences in actual and predicted values.

These differences are not surprising because both SARIMAX and LSTM are based on the past behavior to predict a future outcome instead of reproducing the data. The SARIMAX was found to perform well when dealing with short-term predictions with distinct seasonal variations whereas the LSTM delivered optimal results with more complicated and non-linear relation inside the physiological data. Collectively, they gave a balanced comparison of the traditional statistical and deep learning methods.

The accuracy was also determined by conventional performance measures of Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). The findings showed that the two models obtained low error values with SARIMAX providing stable and explainable forecasts and LSTM providing greater flexibility in the case of irregular data patterns. This affirmed that the predictive model can be used to generate valid and clinical predicted outcomes in real time applications of remote patient monitoring practices.

- **Mean Absolute Error (MAE)**

MAE provides a straightforward and easily interpretable measure of the average magnitude of the errors, expressed in the same units as the observed data (e.g., mg/dL). For example, a MAE of 3.78 mg/dL indicates that, on average the model's predictions gets deviate by approximately 3.78 mg/dL from the actual glucose readings. Many studies show that it is necessary to understand or observe that how deep learning model predict blood glucose [75]. It is less sensitive against surges or external parameters, providing a more stable average error, a point often discussed in the context of precise, personalized forecasting platforms [76].

For Example, If actual values were [100, 110] mg/dL and predictions were [105, 105] mg/dL,

Error 1: $|100-105|=5$

Error 2: $|110-105|=5$

MAE = $(5+5)/2=5.0$ mg/dL

- **Root Mean Square Error (RMSE):**

RMSE is particularly vital in medical scenarios because due to the squaring of each error value produce higher penalty on larger errors. This characteristic makes RMSE highly sensitive to critical prediction failures, such as missing a severe hypoglycemic or hyperglycemic event, which are paramount concerns in diabetes management [77].

- For Example, Using the same actual values [100, 110] mg/dL and predictions [105, 105] mg/dL,

Error 1 squared: $(100-105)^2=(-5)^2=25$

Error 2 squared: $(110-105)^2=(5)^2 = 25$

RMSE = $\sqrt{(25+25)/(2)}$ =5.0 mg/dL

4.4.3 Alert Mechanism Testing (AWS Lambda & SNS)

The alert mechanism testing was necessary to ensure that the system had the ability to identify anomalies and send real-time notifications. Once the simulated data exceeded specific thresholds (like 150mg/dL with blood glucose or 120bpm with heart rate), the system would be analyzed to test that an AWS Lambda function was called to process the data and AWS SNS sent an alert. Breach of

thresholds were put into the data in every test case in order to simulate emergency situations. In the sudden increase of blood glucose up to 180mg/dl or sharp drop of heart rate to 45bpm triggered an artificial alarm. The time taken between when the threshold was exceeded till the email notification was received through SNS was used to determine the response time of the system. The system was also evaluated to make sure that a series of alerts were possible to be raised where necessary and the alerts were designed to be presented in a readable manner with such relevant patient data as timestamp, metric and predictive value.

4.4.4 Limitations of Current Testing

The lesson learned with the simulation was useful but had its own flaws with the use of data whose authenticity is not valid and the lack of real sensors. Though the simulated data were realistic, they are not the same noise, jolts and errors as are found in actual sensor measurements in reality. As a result, the current testing stage was not dedicated to the problem of sensor calibration, noise filtering in real-time or quality of data that might be experienced during real sensor input. In addition, the simulated condition was not in a real-time setting and did not consider the possibility of delays when using cloud-based systems on large scale. Timeliness of alerts may be compromised in practice as a result of network latency and constraint on cloud resources, particularly when large volumes of data are needed to be processed. These will be considered in the next phase at which time real sensor integration and real system performance will be discussed.

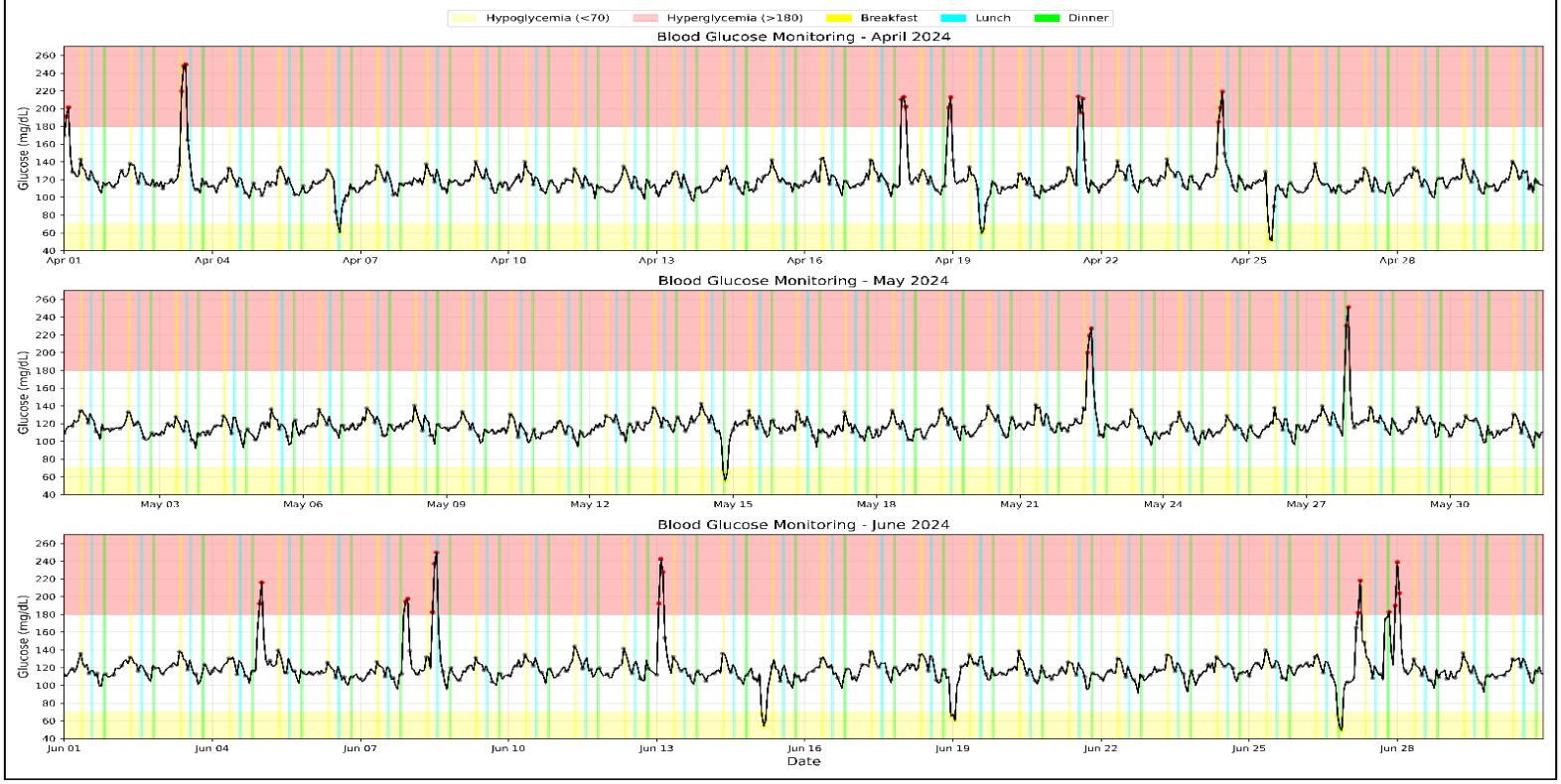
4.5 Accuracy Comparison of SARIMAX & LSTM on Simulated Data

In order to compare the performance of forecasting of the two models, the same simulated datasets of heart rate and blood glucose were applied to both SARIMAX and LSTM models. The two models were trained and tested under the same conditions to be able to compare. SARIMAX was useful in capturing short term linear trends and seasonality and LSTM showed strength in terms of capturing complex nonlinear relationships and long-term temporal relationships. The forecasts were compared after which standard evaluation measures like Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) were used to assess the results. The comparative analysis showed that, even though SARIMAX gave stable and understandable results, the LSTM had a little higher accuracy and was more adaptable to sudden changes, which means it may be applicable to the medical forecasting systems in real-time. Before the physical deployment of sensors to measure vital signs of these locations, the functionality and effectiveness of the proposed hybrid remote patient monitoring (RPM) architecture were assessed through the creation of simulated datasets of the anticipated heart rate and current blood glucose trends. These synthetic data capture real world physiological differences, anomalies and noise patterns that are normally found in the clinical monitoring. Synthetic data in turn made it possible to extensively test the time-series forecasting pipeline, alerts and responsiveness of the system in a controlled environment. This markets section is a critical analysis on the way the system behaves in simulation to check its predictive accuracy, ability to detect anomalies and performance in handling data.

4.5.1 Simulated Blood Glucose Data

The mock heart rate data, generated over seven days, mimics normal physiological changes alongside intentional anomalies for evaluating system performance.


Hypoglycemia (<70)
Hyperglycemia (>180)
Breakfast
Lunch
Dinner
Blood Glucose Monitoring - January 2024
Blood Glucose Monitoring - February 2024
Blood Glucose Monitoring - March 2024
Glucose (mg/dL)
Date


Hypoglycemia (<70)
Hyperglycemia (>180)
Breakfast
Lunch
Dinner
Blood Glucose Monitoring - April 2024
Blood Glucose Monitoring - May 2024
Blood Glucose Monitoring - June 2024
Glucose (mg/dL)
Date

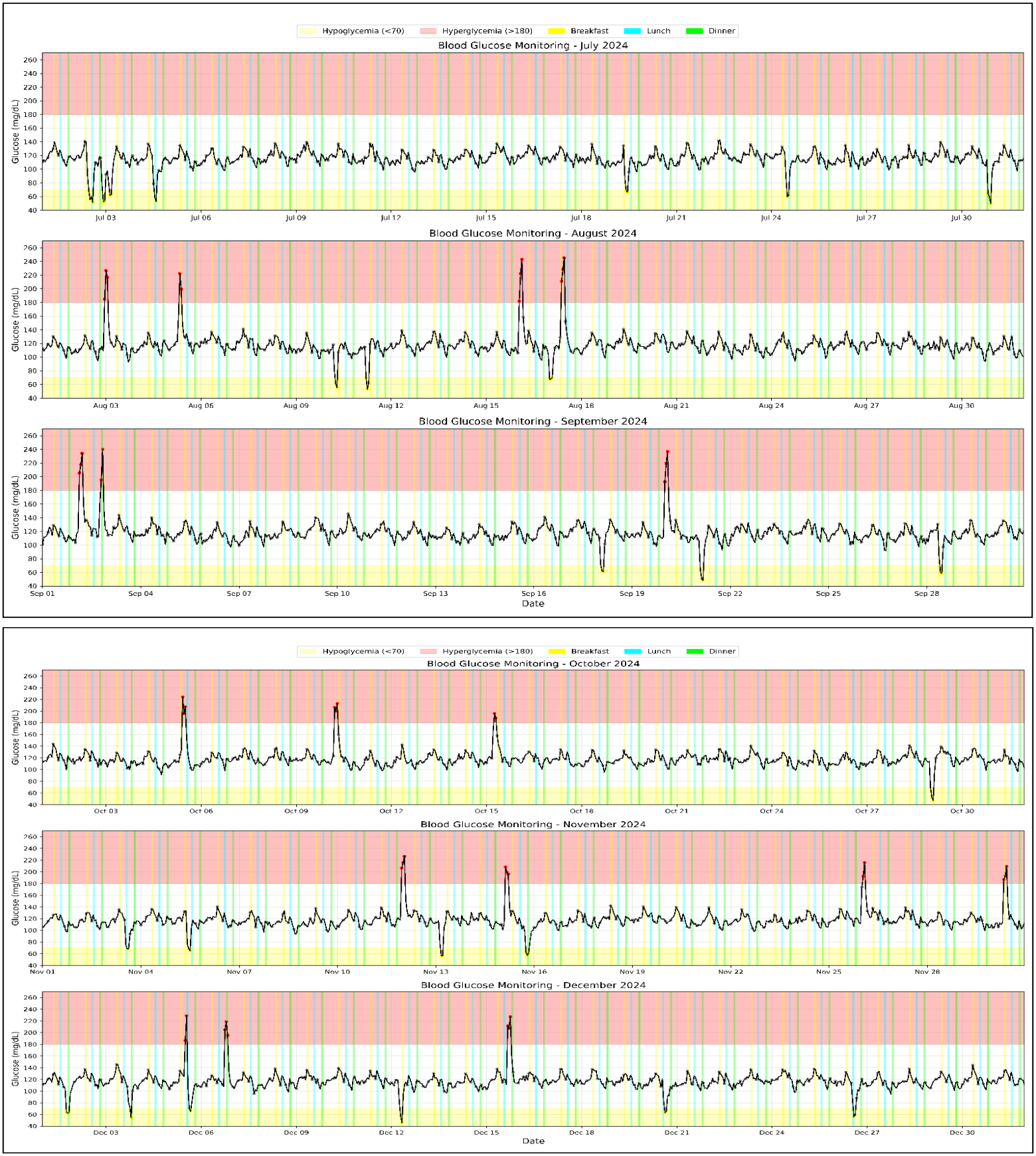


Figure 4.1: Data Plots Showing Blood Glucose Data JAN – DEC 202

It includes plotted figures representing the simulated data, i.e. glucose-monitoring data, in the time period between January and December 2024 with Python script being put to work in a Visual Studio (or VS Code) development environment. The figures illustrate change in systolic blood glucose levels over time in mg/dl with background coloring changing to represent the hypoglycemic (less than 70mgdl, yellow) and hyperglycemic (greater than 180mgdl, red) ranges. Mealtimes are marked with the help of vertically yellow colored bands, blue one being breakfast, green one lunch, and red/pink one dinner. The resulting time series have predictable postprandial peaks which frequently exceed hyperglycemic thresholds, with interspersions of hypoglycemic troughs, which is typical of a synthetically created, perhaps improperly controlled diabetes model used to instruct, or validate.

- **Python Script**

See code **"A-1 Simulated Blood Glucose Data"** in APPENDIX A

4.5.2 Implementation of SARIMAX on Different Sized Data Sets

After the data-sets had been prepared, each of the three data-set lengths was then fitted to the SARIMAX model. The initial test used one-month series (January1- January 31, 2024), which had a narrow history, resulting in a fast convergence of the model though with less predictable results. The second experiment extended the training period to 6 months (January 1 -June 30, 2024) and thus introduced a larger history that increased the stability and accuracy of the forecasts. The last experiment utilized the complete twelve months' series (January1- December 31, 2024), allowing the model to learn the seasonal dynamics long run at the expense of more training time and computation time. Such repeated expansion of time horizons guaranteed the methodical evaluation of SARIMAX behavior by short, medium- and long-term historical data.

4.5.2.1 Results of SARIMAX for Small Data Set

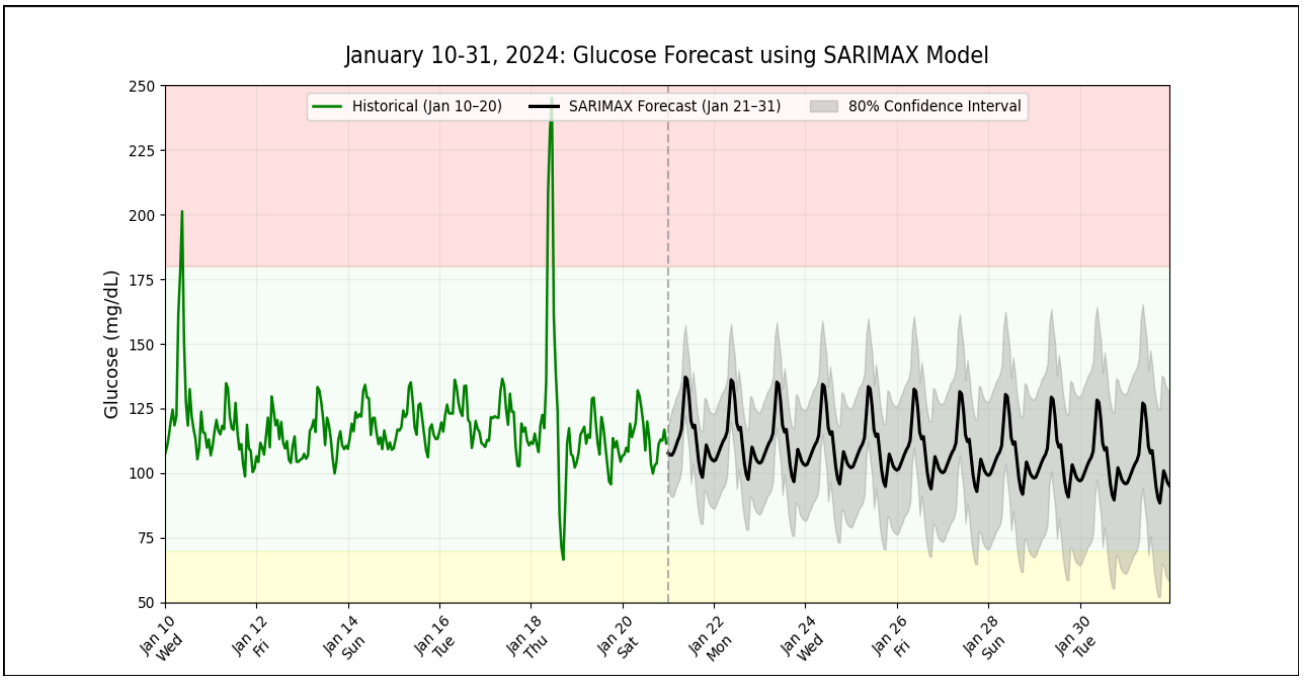


Figure 4.2: Plot Showing SARIMAX Forecasting Results On 1 Month Training Data

The plot shows how SARIMAX model declared the blood glucose level of January 21-31, 2024, using one month of data obtained during the period of January 2024. The historical part (January120), which can be regarded as a loud green line, shows training data that is used to create the model and a rough spike of hyperglycemia on January18. The forecast section, represented by a smooth black line past the gray dashed divider, represents the general day-long glucose pattern with the sharp spikes in the response of the training data left out. Colors are used to indicate the glucose conditions: yellow (case of hypoglycemia), green (normal glucose level), and red (hyperglycemia). Altogether, the prediction is within clinically reasonable standards, which proves that the SARIMAX model has been able to recreate long-term glucose patterns based on one-month of training data, as well as remove the noise of the short terms.

.

4.5.2.2 Results of SARIMAX for Medium Sized Data Set

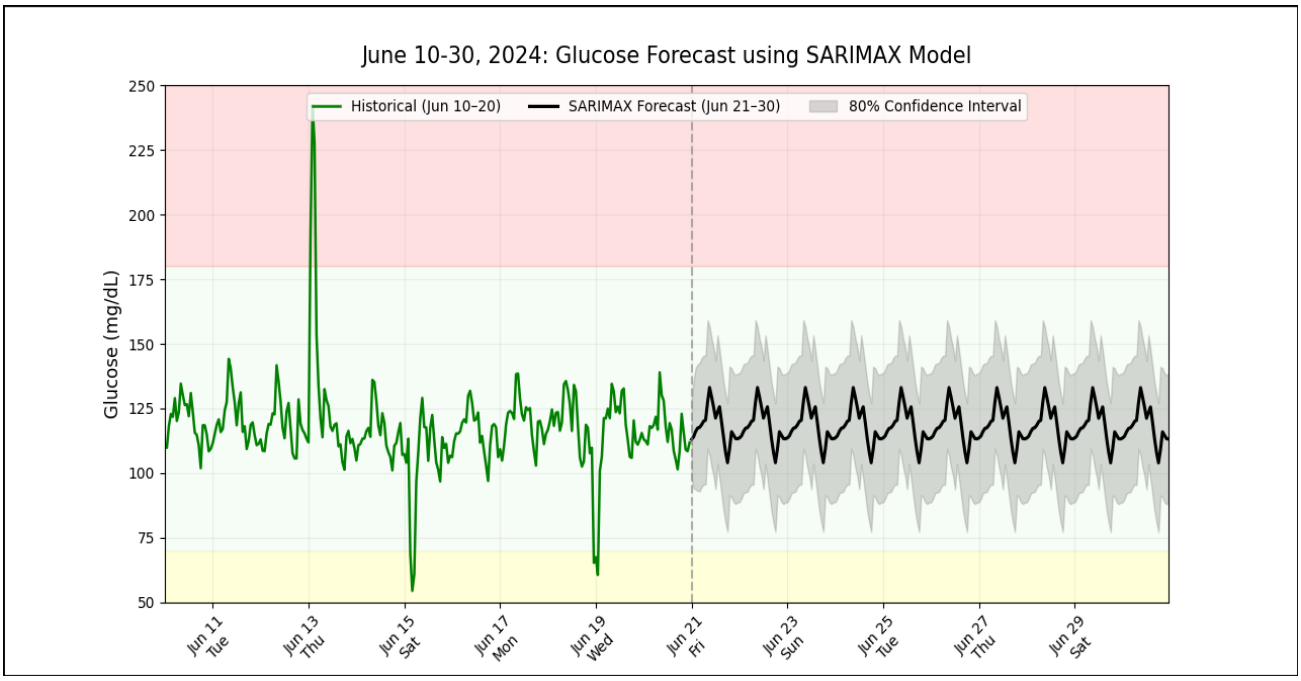


Figure 4.3: Plot Showing SARIMAX Forecasting Results On 6 Months Training Data

Figure 4.3 plot shows the prediction capabilities of a SARIMAX model used on a medium size data base of blood-glucose measurements. The model had six months of historical data as similar input between 1 January and 30 June and the model was then given the responsibility to predict glucose levels between the horizon of 21 June and 30 June. Analysis of the historical part of the plot demonstrates a very noisy and irregular shaped pattern, with a few conspicuous spikes and dips, most importantly, there is a significant hyperglycemic spike on 13 June and a hypoglycemic dip on 15 June.

On the contrary, the segment being predicted (a smooth black line) has a more stable and regular cyclic trend. This fact implies that the SARIMAX model was effective in capturing the underlying seasonality and overall time behavior of the glucose fluctuations in the data. Nevertheless, the seamless fluctuation and stochastic noise of the prediction data point to a major weakness of the model the decreased ability to predict the sudden and temporary anomaly or the sudden physiological alteration. In general, the findings prove that SARIMAX works, but it can be complemented by more reactive models including long short-term memory (LSTM) networks to respond more to the immediate changes in physiological conditions

4.5.2.3 Results of SARIMAX for Large-Sized Data Set

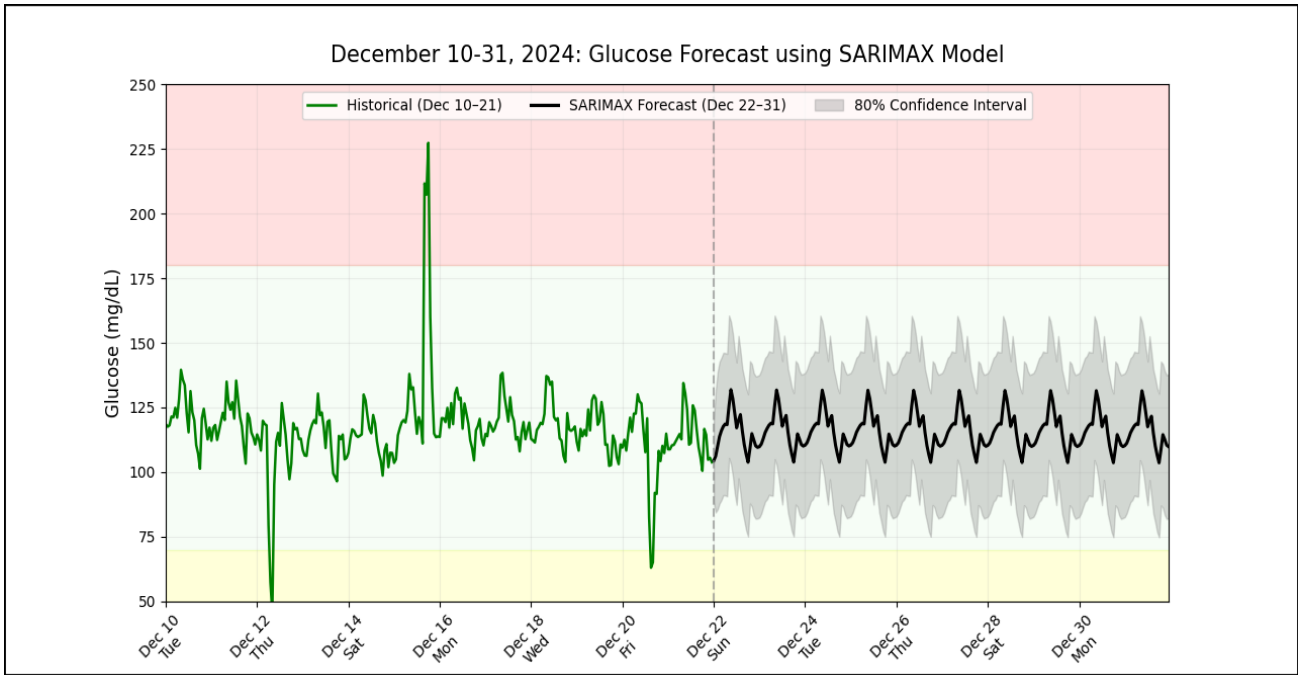


Figure 4.4: Plot Showing SARIMAX Forecasting Results On 12 Months Training Data

The plot in figure 4.4 illustrates the SARIMAX model's forecast of blood glucose levels for the period from December 22 to 31, 2024. The plot is visually divided by a vertical dashed gray line, separating the historical data (December 10–21) from the forecasted period. The historical portion, depicted as a green line, exhibits typical fluctuations found in real-world glucose readings, including both gradual variations and sharp, erratic changes. A prominent hyperglycemic spike on December 15 and a sharp hypoglycemic drop on December 13 & 20, which reflects that unpredictable variations occur due to some exog variables like medication or physical activity. This historical data served as the training input for the SARIMAX model, enables it to learn the seasonal dependencies impacting on glucose fluctuations.

On the other hand ,the forecasted region represented by black line at the right of the plot, shows a more regular and cyclical rhythm.

- **Python Script**

See code **“A2 - SARIMAX Forecasting Blood Glucose Data”** in APPENDIX A

4.5.3 Comparative Analysis of SARIMAX Forecasted & Historical Data

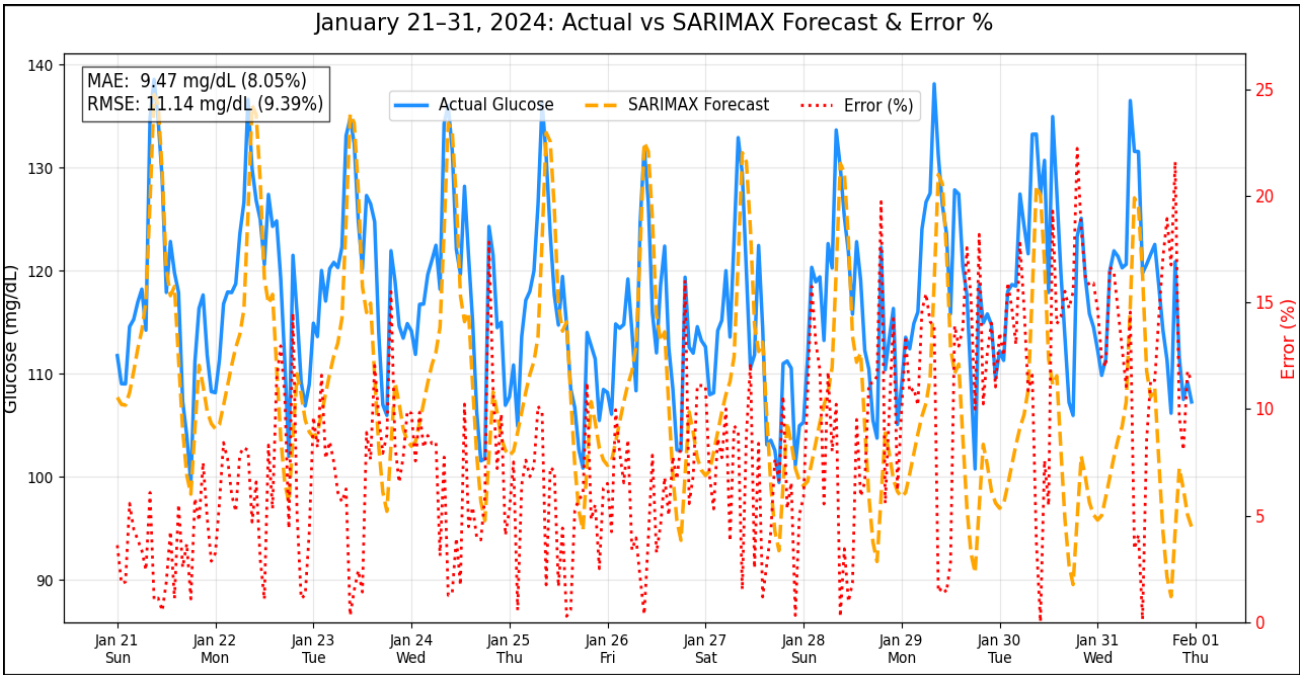


Figure 4.5: Plot Showing Actual & Historical Data with% Error for Small Data Set

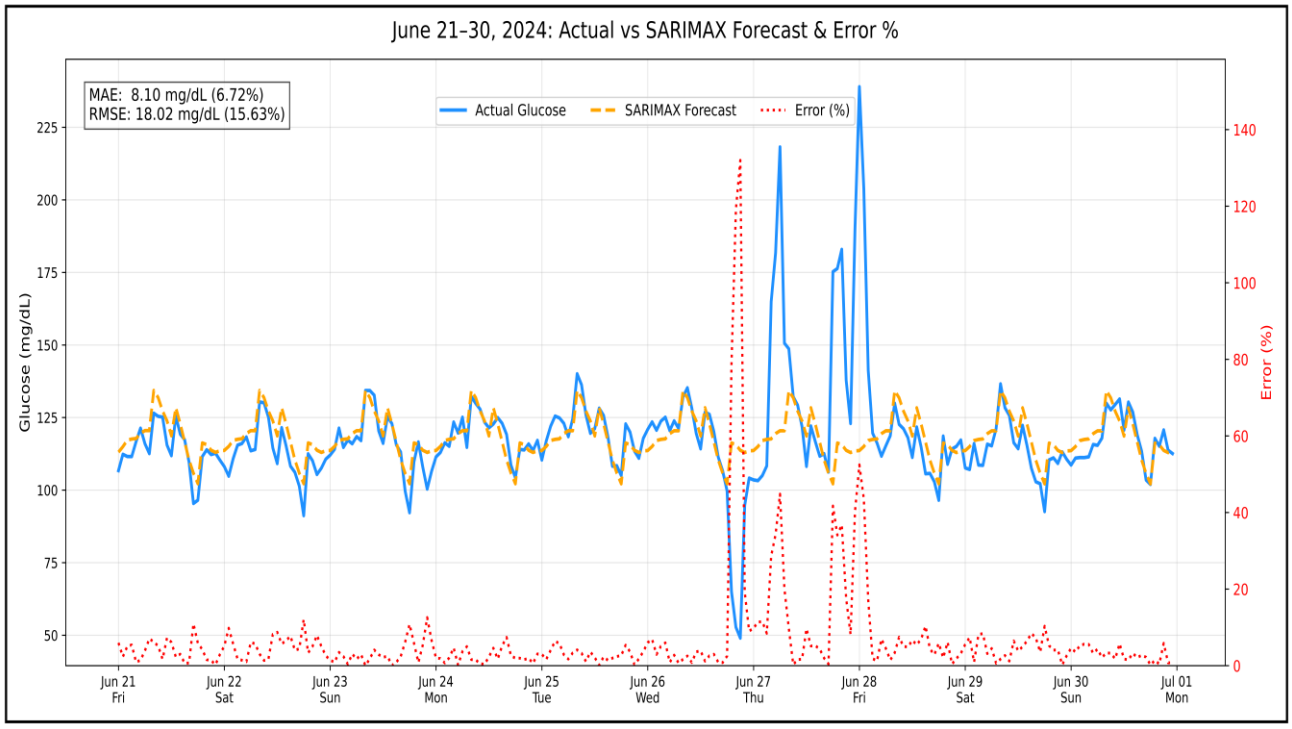


Figure 4.6: Plot Showing Actual & Historical Data with% Error for Medium Data Set

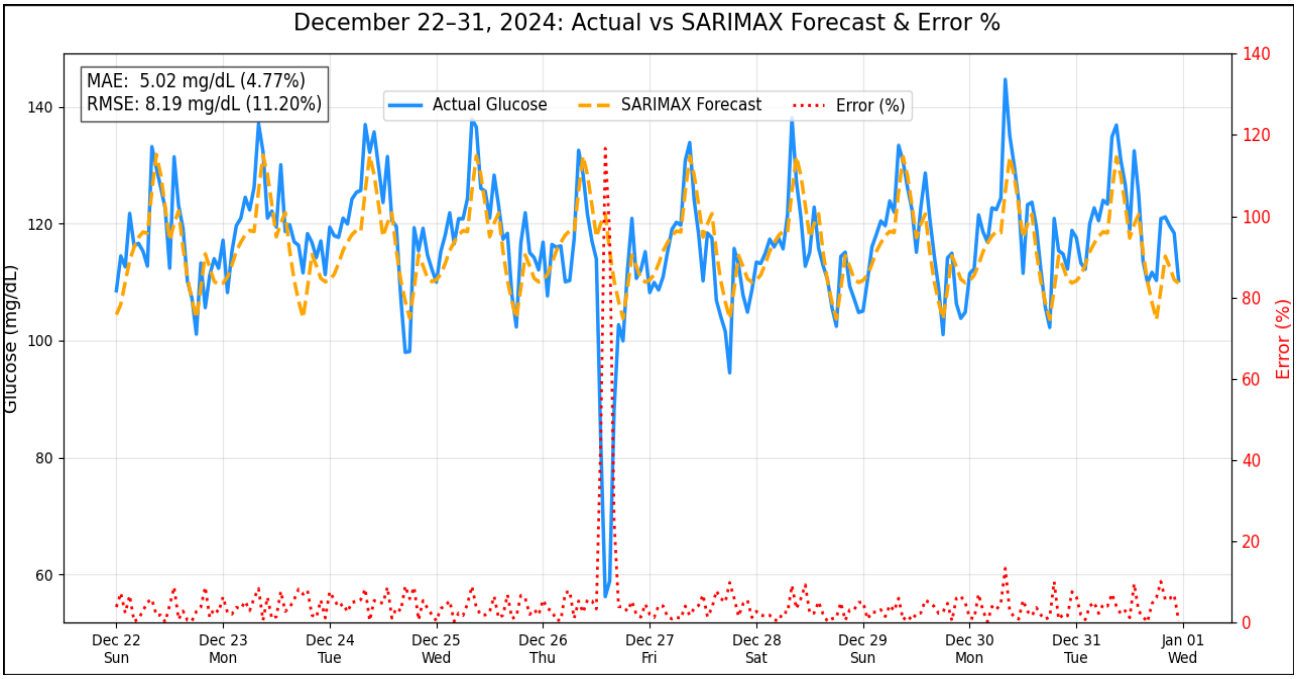


Figure 4.7: Plot Showing Actual & Historical Data with % Error for Large Data Set

The three plots in figure 4.5, 4.6 & 4.7 clearly demonstrate how training data size impacts SARIMAX forecasting accuracy. In the first plot (January 2024), the model trained on a small one-month dataset performs poorly, missing major peaks and troughs, resulting in a high MAE of 8.05%. In the second plot (June 2024), a larger dataset enables the model to learn recurring glucose patterns, producing smoother forecasts that better follow actual trends and reduce the MAE to 7.02%. Finally, in the December 2024 plot, the model trained on the most extensive dataset achieves its best performance, closely matching real fluctuations with minimal deviation and the lowest MAE of 4.77%. This progression confirms that larger, richer datasets significantly improve the SARIMAX model's predictive accuracy and reliability.

- **Python Script**

Please refer to code **"A3-Historical & SARIMAX Forecasted Data Comparison"** in APPENDIX A

4.5.4 Implementation of LSTM On Different Sized Data Sets

Same like SARIMAX, the LSTM model was implemented separately on each of the three dataset lengths. The 1-month dataset (January 1st – January 31st, 2024) was first used to test the model with a limited history, resulting in quick training but less stable forecasts. Next, the 6-month dataset (January 1st –June 30th, 2024) was applied, providing a larger history that improved stability and accuracy. Finally, the 12-month dataset (January 1st –December 31st, 2024) was used, allowing the model to capture long-term seasonal patterns, though it required higher training time and computational resources. This step ensured a systematic evaluation of LSTM performance under short-, medium-, and long-term historical records. Please refer to python scripts below used for implementation of LSTM

4.5.4.1 Results of LSTM for Small Data Set

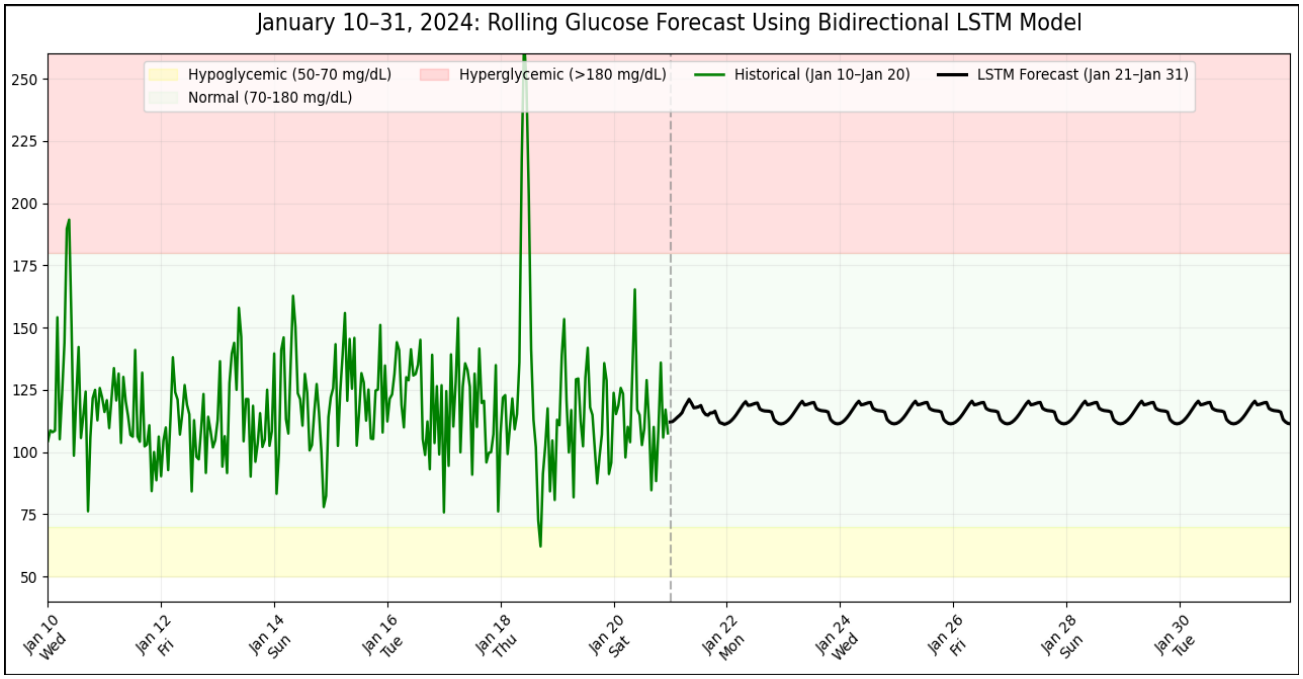


Figure 4.8: Plot Showing LSTM Forecasting Results On 1 Month Training Data

The plot illustrates the bidirectional LSTM model's forecast of blood glucose levels for the period January 10–31, 2024. The initial (point January10-20), depicted as a noisy green line, fits the historical training data that involves natural variations and a high hyperglycemic event on January18. This was the dataset used to train the model on identifying dynamic glucose. The chart includes chromatically-coded bands, i.e. the yellow to signify hypoglycemia, the green to signify the normative range and the red to represent hyperglycemia; grey is a dashed line that indicates the start of the forecasting of the event.

During the period of January 21 -31, the estimated trend, which is represented by a smooth black line, describes the dominating circadian pattern of variations of glucose and suppresses high-frequency fluctuations and individual anomalies evident in the past history series. This result depicts that the bidirectional LSTM is able to capture long amounts of time dependencies and is also capable of replicating more stable glucose patterns, despite its only mild success in closely replicating acute, stochastic swings.

4.5.4.2 Results of LSTM for Medium Data Set

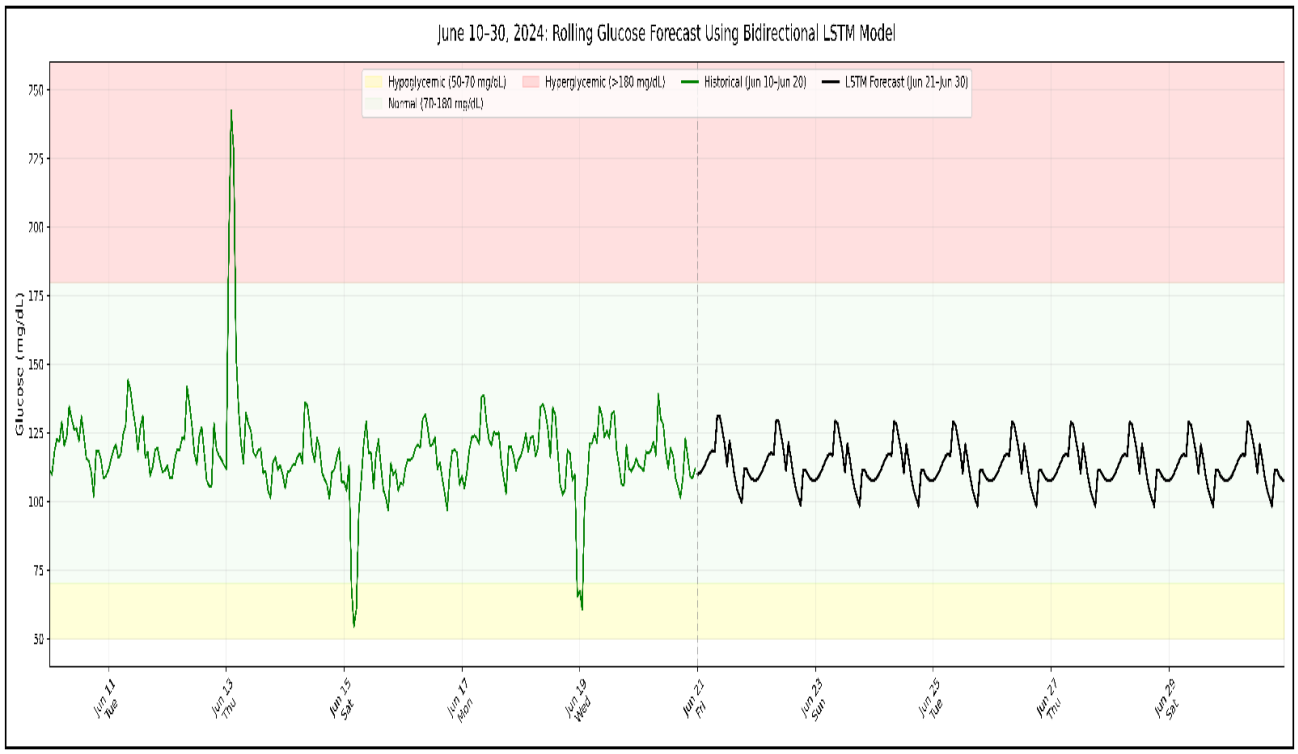


Figure 4.9: Plot Showing LSTM Forecasting Results On 6 Months Training Data

The figure shows how LSTM predicts the levels of blood sugar in the range of 21 30 June 2024, given 10 January to 20 June training data. A gray vertical dashed line is used to mark the historical section (identified by the color green) and the forecasted one (identified by the color black). A considerable hyperglycemic peak on the 13th day of June and a hypoglycemic trough on the 15th day of June characterize the spikes and troughs in the historical data, showing a natural variation.

The predicted segment has a consistent cyclic relation, and it is smooth and stable, which makes it possible to assume that the LSTM was able to learn the underlying daily pattern of glucose variability. However, it does not replicate the random high-frequency noise or sudden spikes thus highlighting the weakness of the model in modeling unexpected irregular occurrences. On the whole, the prediction highlights how effectively the model acquires glucose behavior learning in the long term though at the cost of implementing the overstated fluctuations that are portrayed in actual life.

4.5.4.3 Results of LSTM for Large Data Set

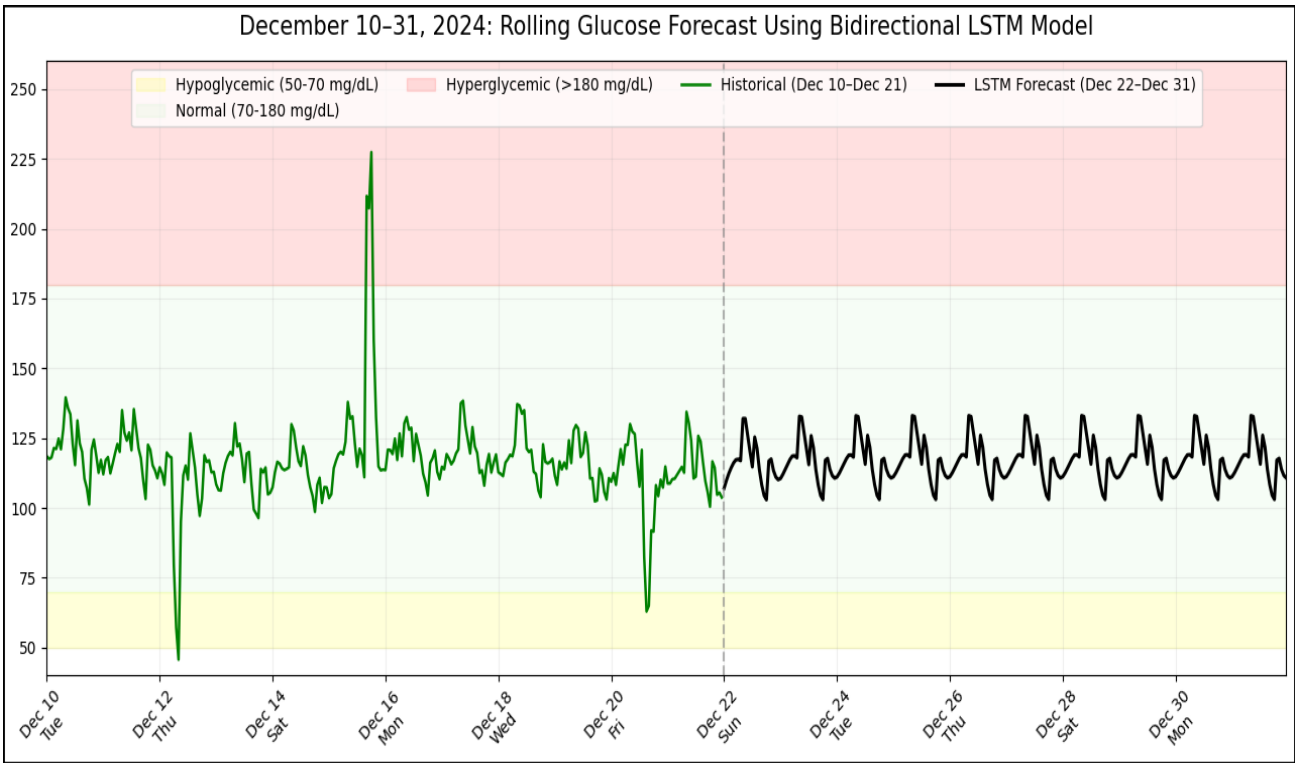


Figure 4.10: Plot Showing LSTM Forecasting Results On 12 Months Training Data

The plot presents the forecast of blood glucose levels from December 10 to December 31, 2024. The plot is divided by a vertical dashed gray line, separating the historical data (December 10–21) and the forecasted period (December 22–31). The historical data, shown in green, displays realistic physiological variability, including a notable hyperglycemic spike on December 15 and a brief hypoglycemic dip on December 20, reflecting the natural fluctuations seen in real-world glucose measurements.

On the right side of the plot the forecasted region black line shows that the trends are captured more accurately as compared to the plot where limited one-month training data was provided. So it shows that accuracy is high in capturing the trends as compared to short data set. So that's why this type of model perform better with larger data sets as compared to the scenario where short or limited data is available

- **Python Script**

Please refer to code "**LSTM Forecasting Blood Glucose Data**" in APPENDIX A

4.5.5 Comparative Analysis of LSTM Forecasted & Historical Data

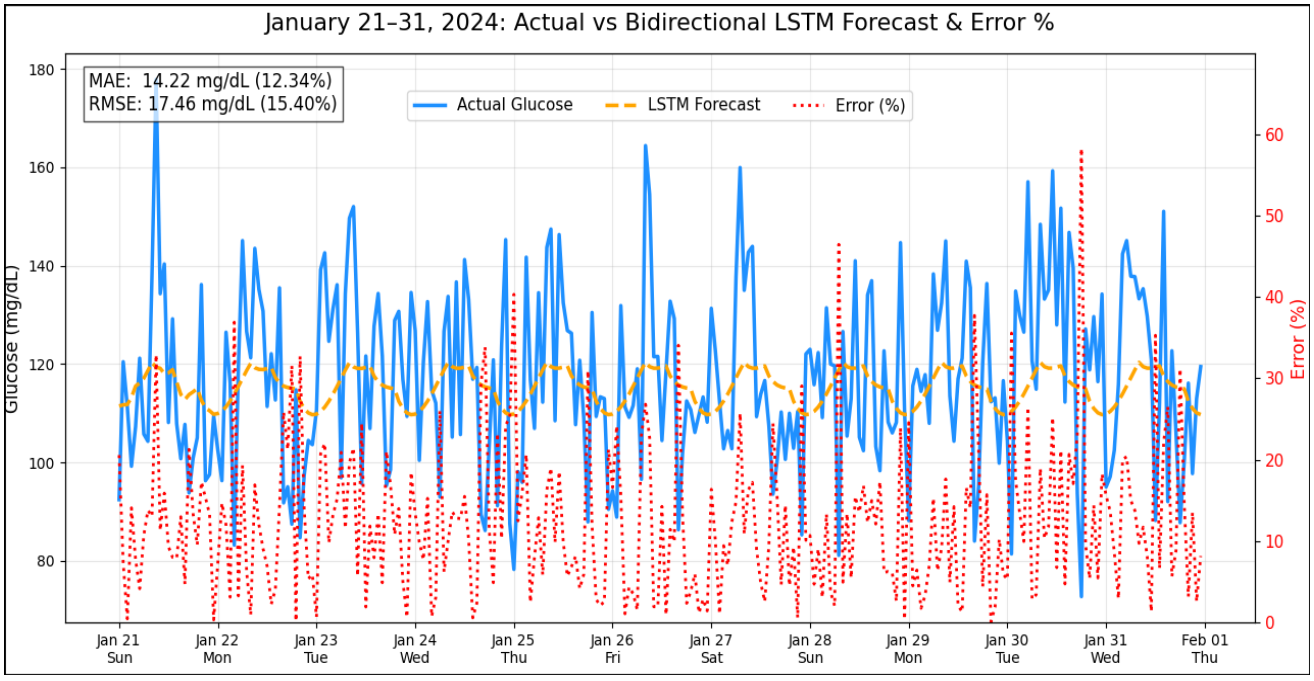


Figure 4.11: Plot Showing Actual & Historical Data with% Error for Small Data Set

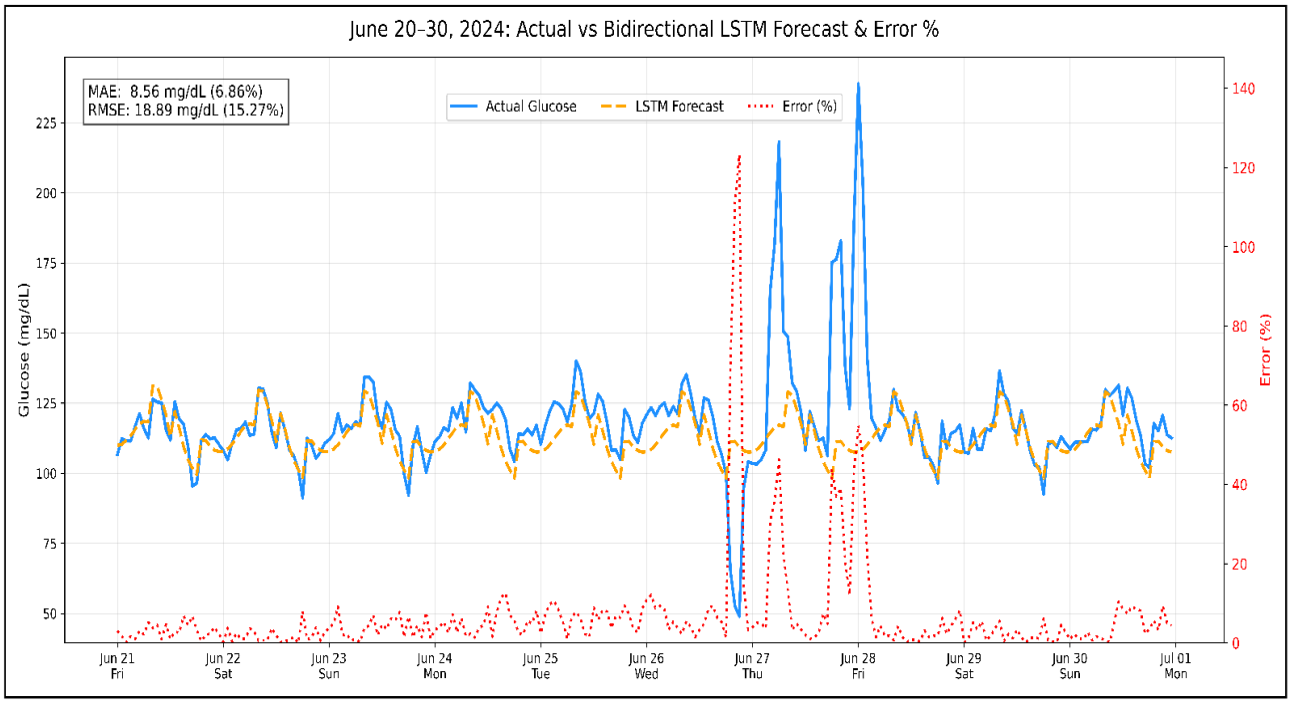


Figure 4.12: Plot Showing Actual & Historical Data with% Error for Medium Data Set

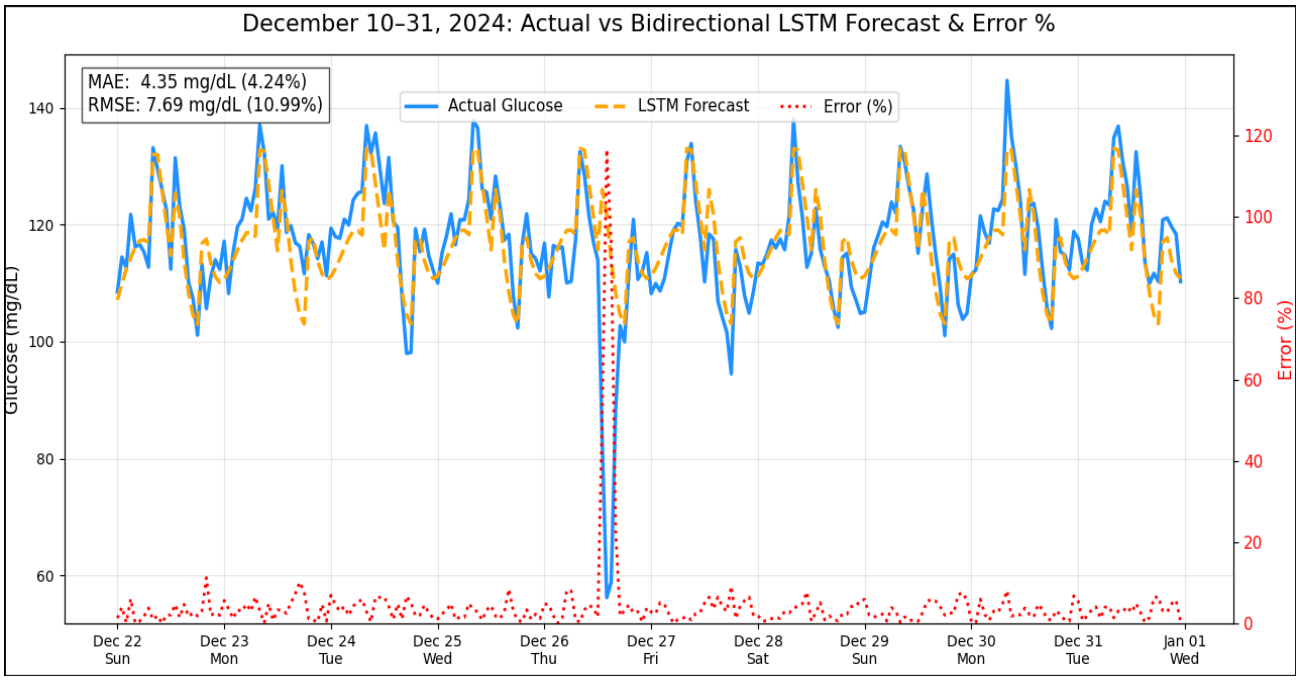


Figure 4.13: Plot Showing Actual & Historical Data with% Error for Large Data Set

In the first case (January 21–31, 2024), the Bidirectional LSTM model was trained on a small, only one-month dataset, that limited its capability to capture the forecasted patterns. Due to the insufficient training data , the model producing a smooth and almost a straight line with very little fluctuations that

overlooked the natural fluctuations or patter, spikes, and noise typical in real physiological signals. So, the Mean Absolute Error (MAE) is 12.34%, reflecting the model's weak inability to represent short-term variations or irregular glucose changes present in the training data.

On the other hand , the forecasts (June 20–30 and December 10–31, 2024) demonstrate progressive improvement as the training dataset size increased. With the addition of longer-term data, the model s identified recurring temporal dependencies and daily glucose cycles, resulting in smoother yet more accurate trend predictions. The June model achieved an MAE of 6.86%, showing a strong reduction in forecasting error, while the December model that was trained on the largest dataset having maximum available values produced highly realistic forecasts with an MAE drops below to 4.24%.

- **Python Script**

Please refer to code **"A5-Historical & LSTM Forecasted Data Comparison"** in APPENDIX A

4.5.6 Comparative Analysis SARIMAX VS LSTM in Terms of Accuracy

The comparative analysis between SARIMAX and LSTM models was performed using Python scripts to evaluate their forecasting performance based on previously generated results. Existing forecasting outputs are utilized instead of creating newer ones from earlier experiments on blood glucose prediction across varying dataset lengths.. To conduct the analysis, the Python environment was used to import and process both models' outputs, followed by the computation of key accuracy metrics Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). These metrics offered a clear quantitative comparison of predictive precision. Visualization using Matplotlib and Pandas further illustrated performance differences across data scales, showing that SARIMAX performed better with high accuracy and lower error with smaller data. However the LSTM accuracy enhanced when lagrer data was provided for training the model and forecasting.

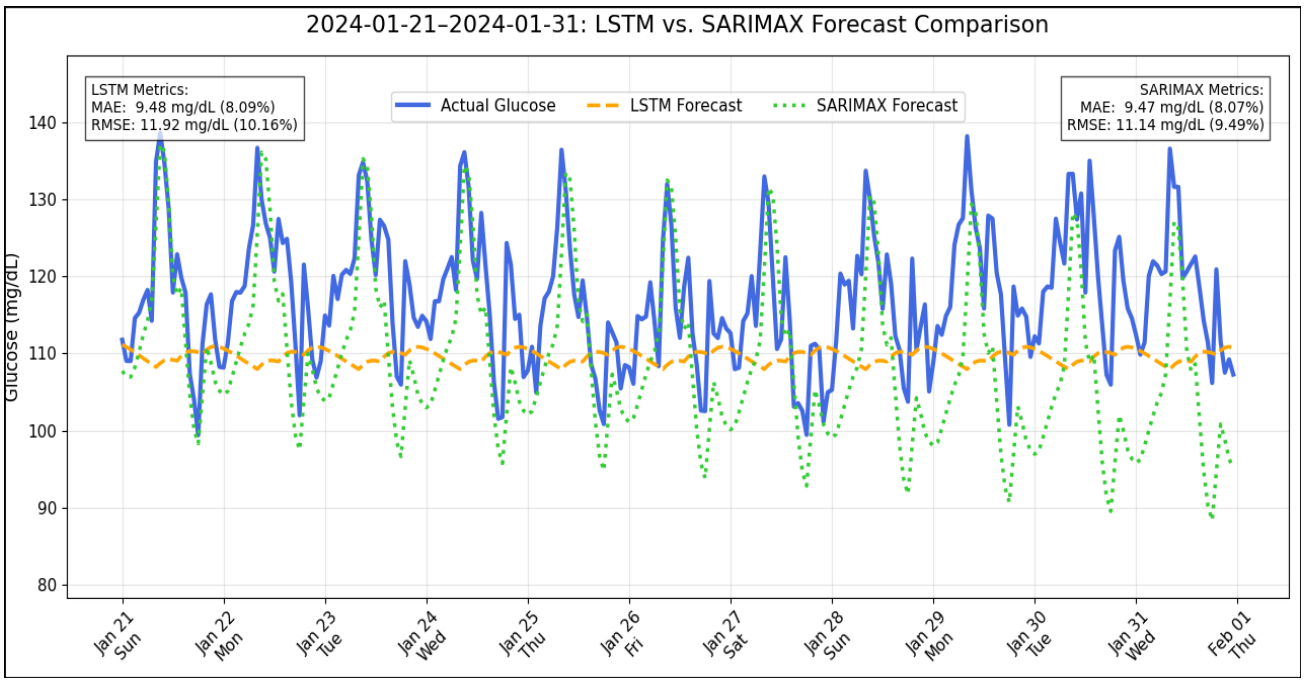


Figure 4.14: Comparison of SARIMAX and LSTM Forecasts Using Identical One-Month Training Data (1st January 1–31st January, 2024).

Figure 4.14 compares blood glucose forecasts for January 2024 using SARIMAX and LSTM models. SARIMAX captures daily fluctuations more effectively as compared to LSTM that is unable to capture the real trend. With lower MAE (8.07%) and RMSE (9.49%) than LSTM (MAE 8.09%, RMSE 10.16%), SARIMAX shows slightly better accuracy for this period.

.

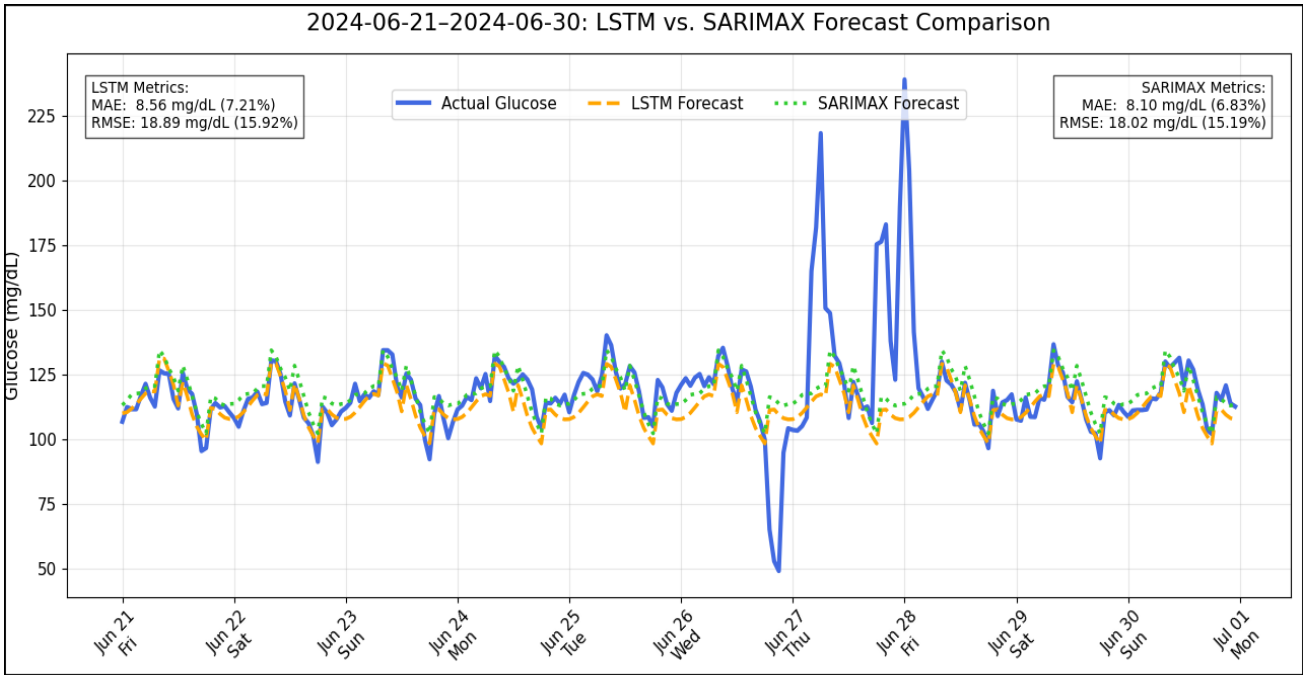


Figure 4.15: Comparison of SARIMAX and LSTM Forecasts Using Identical 6-Months Training Data (1st January – 30th June, 2024).

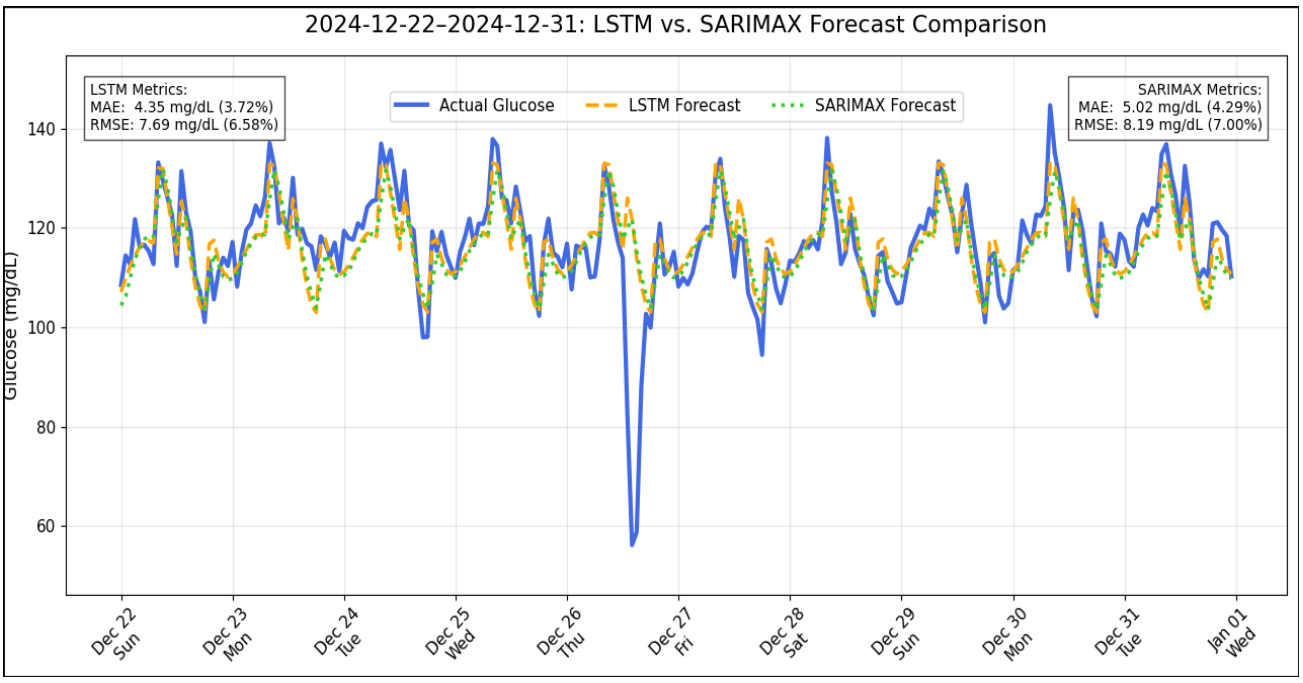


Figure 4.16: Comparison of SARIMAX and LSTM Forecasts Using Identical 12-Months Training Data (1st January – 31st December, 2024).

The comparison between the LSTM and SARIMAX models for blood glucose forecasting highlights a clear difference in how each handles data volume. In Figure 4.15, where only limited historical data for 6 months were available, both models struggled to capture sudden glucose fluctuations, though SARIMAX showed a little bit of better accuracy with a lower average error. While the Figure 4.16 shows training of model on a full year data the huge enough for the LSTM model so it performed better by producing lower error values and tracking actual glucose levels more closely. This indicates that while SARIMAX performs reasonably well with smaller datasets, the LSTM model demonstrates higher accuracy when trained on larger data volumes, making it more suitable for long-term glucose forecasting.

- **Python Script**

Please refer to code **"A6-Performance & Accuracy Comparison SARIMAX & LSTM"** in APPENDIX A

4.5.6.1 Forecasting Accuracy Comparison SARIMAX & LSTM

When the performance of SARIMAX and LSTM models in terms of forecasting performance are evaluated, a clear trend develops which is exceptionally dependent on the amount of training data provided. In the test, with only one month of data, SARIMAX model will be more accurate with lower error values. This is due to the fact that SARIMAX has the ability to quickly identify and utilize its basic statistical features, e.g. seasonality and patterns, which are critical in sound forecasting especially where there is limited data in the past.

In their turn, the larger the training set, the more time (six and twelve months) the better the LSTM model performs and eventually, of course, exceeds the performance of SARIMAX. This fact highlights the main benefit of deep learning structures, that they are able to learn non-linear relationships, which are not captured by traditional statistical models, given a sufficiently large set of data. The ability of the LSTM model to use the additional information contained in the large amounts of data are converted into more accurate predictive results.

4.6 Forecasting Models Execution On Microcontroller

Two types scripts were executed on microcontroller for same BG Data Set. SARIAMX and LSTM.

4.6.1 SARIMAX Python Based Script Execution

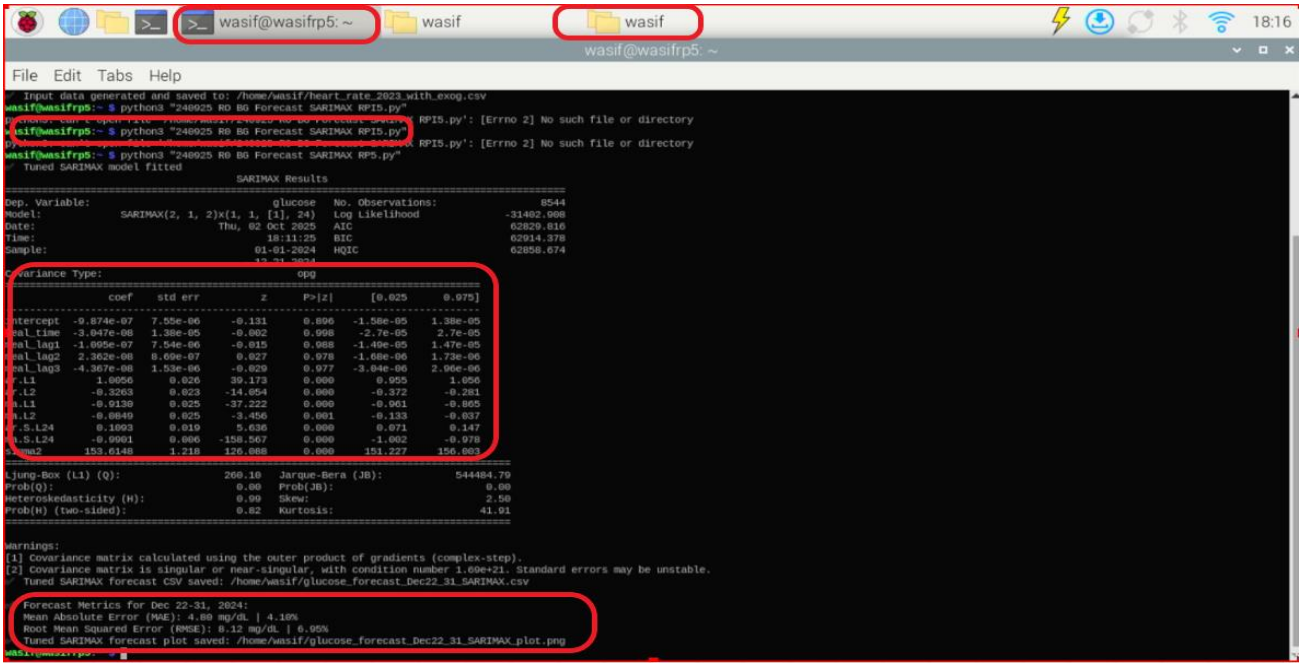


Figure 4.17: Raspberry PI 5 CLI Showing Python Script Implementation



Figure 4.18: Execution of Python Script in Raspberry PI 5 Showing Results in Plot

The provided CLI and GUI verifies the successful execution and validation of the Python forecasting script on the Raspberry Pi that fulfils the core objective of the project. The script efficiently runs the SARIMAX (Seasonal Auto Regressive Integrated Moving Average with Exogenous Repressors) model, which is specifically configured to account for both short-term dependencies and the pronounced 24-hour seasonality characteristic of Blood Glucose (BG) data, along with relevant physiological exogenous variables. The CLI summary provides the critical performance validation, demonstrating a statistically robust outcome with a Mean Absolute Error (MAE) of just 4.80 mg/dL (4.10% tolerance), verifying the model's high predictive accuracy on the edge device. Concurrently, the GUI screenshot visually confirms the script's final output the glucose_forecast_Dec22_31_SARIMAX_plot.png showing the seamless transition from historical blood glucose data to the predicted future curve, thus establishing the system's operational viability for continuous, localized, and highly accurate predictive monitoring.

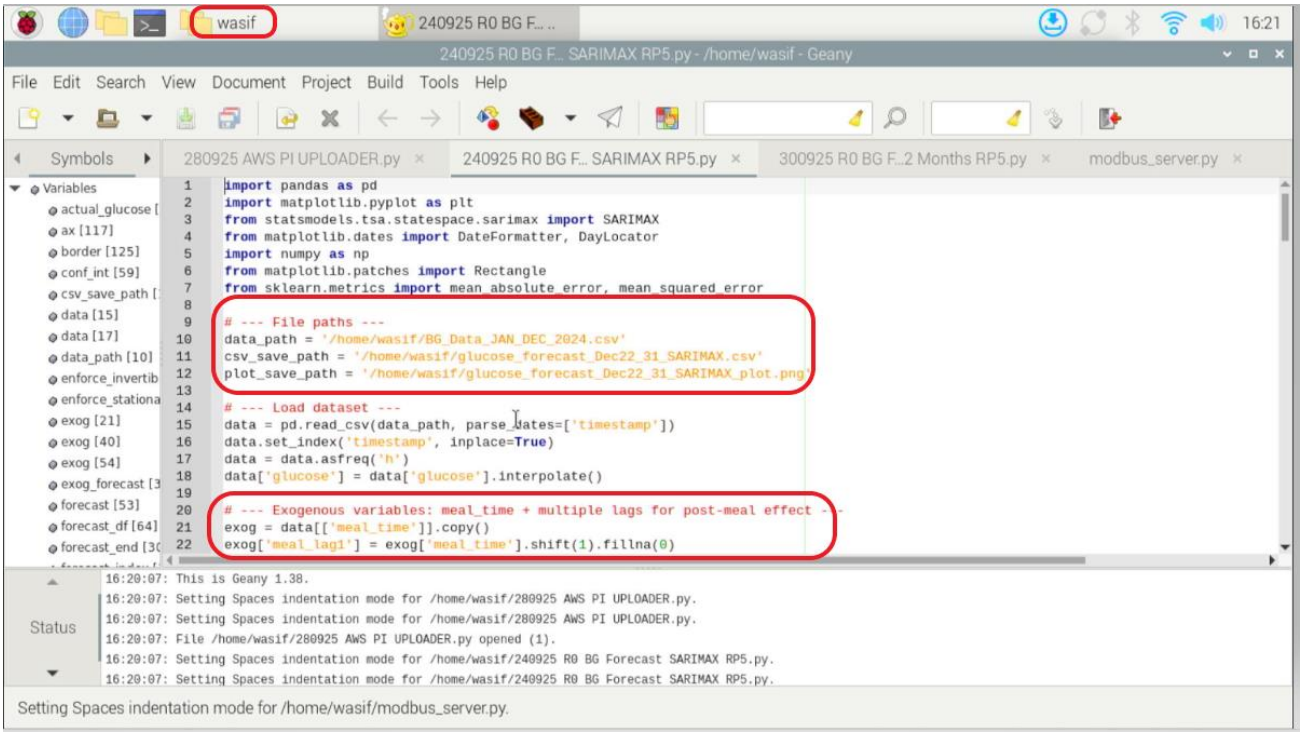


Figure 4.19: Execution of SARIMAX Python Script Showing Results in Plot

The above figures show the execution of python script executed on micro microcontroller.

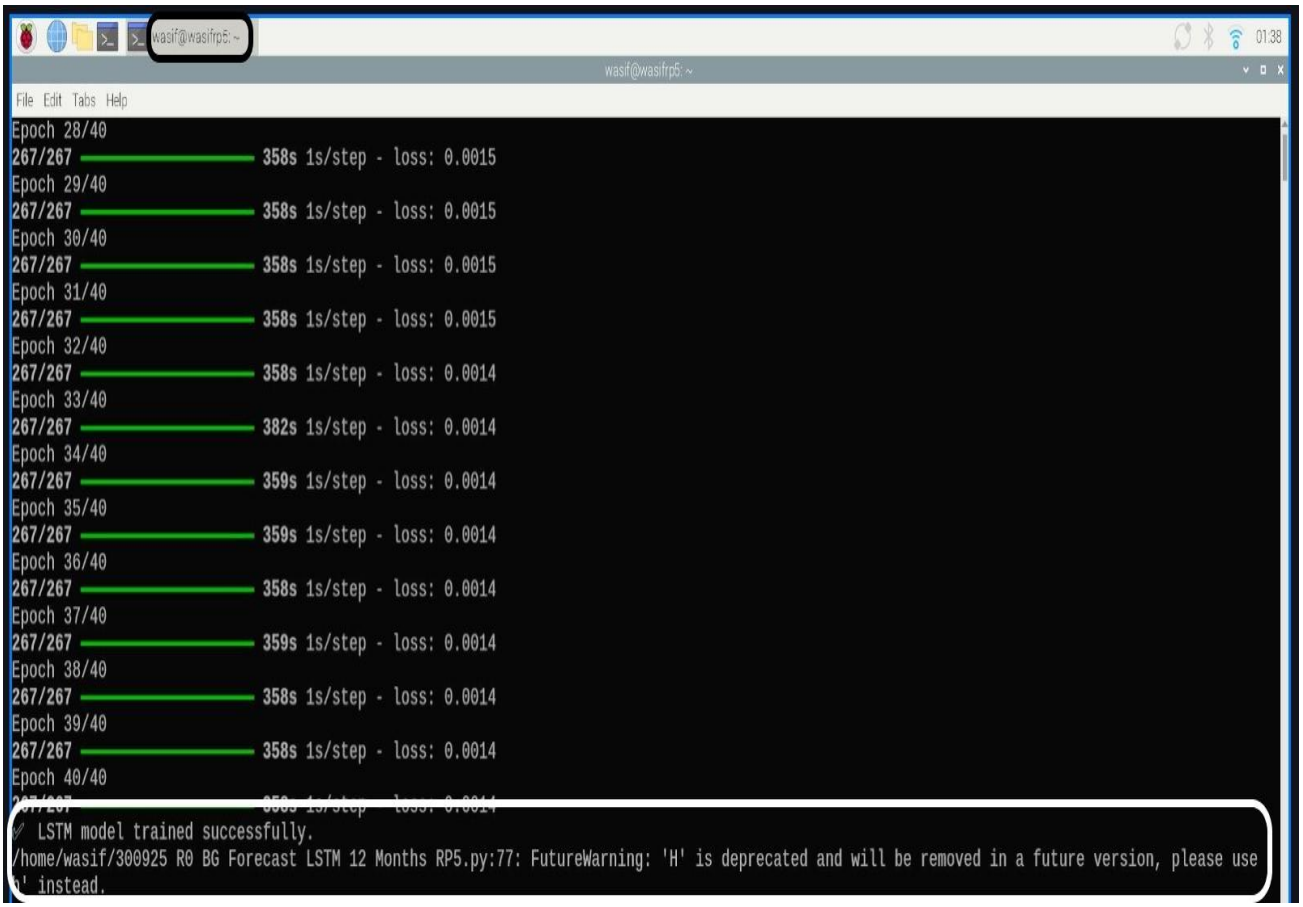


Figure 4.20: Execution of LSTM Python Script in CLI of Controller



Figure 4.21: Execution of LSTM Python Script On GUI of Controller

Above figures show the execution of python script for LSTM. Overall, it involves very high processing and took about 5 Hours to complete the script. The command line interface of the controller shows the completion of execution at the last encircled lines, while at the same time the GUI displays the plot. As well as it also saves the CSV files both on Visual Studio and Raspberry PI.

4.7 Alert Generation and AWS Integration Results

To demonstrate automated alerting in a real-time health monitoring environment, AWS Lambda and Simple Notification Service (SNS) were integrated into the hybrid RPM . This section shows how these cloud services are used to monitor predicted blood glucose levels and simulate timely notifications when critical thresholds were breached.

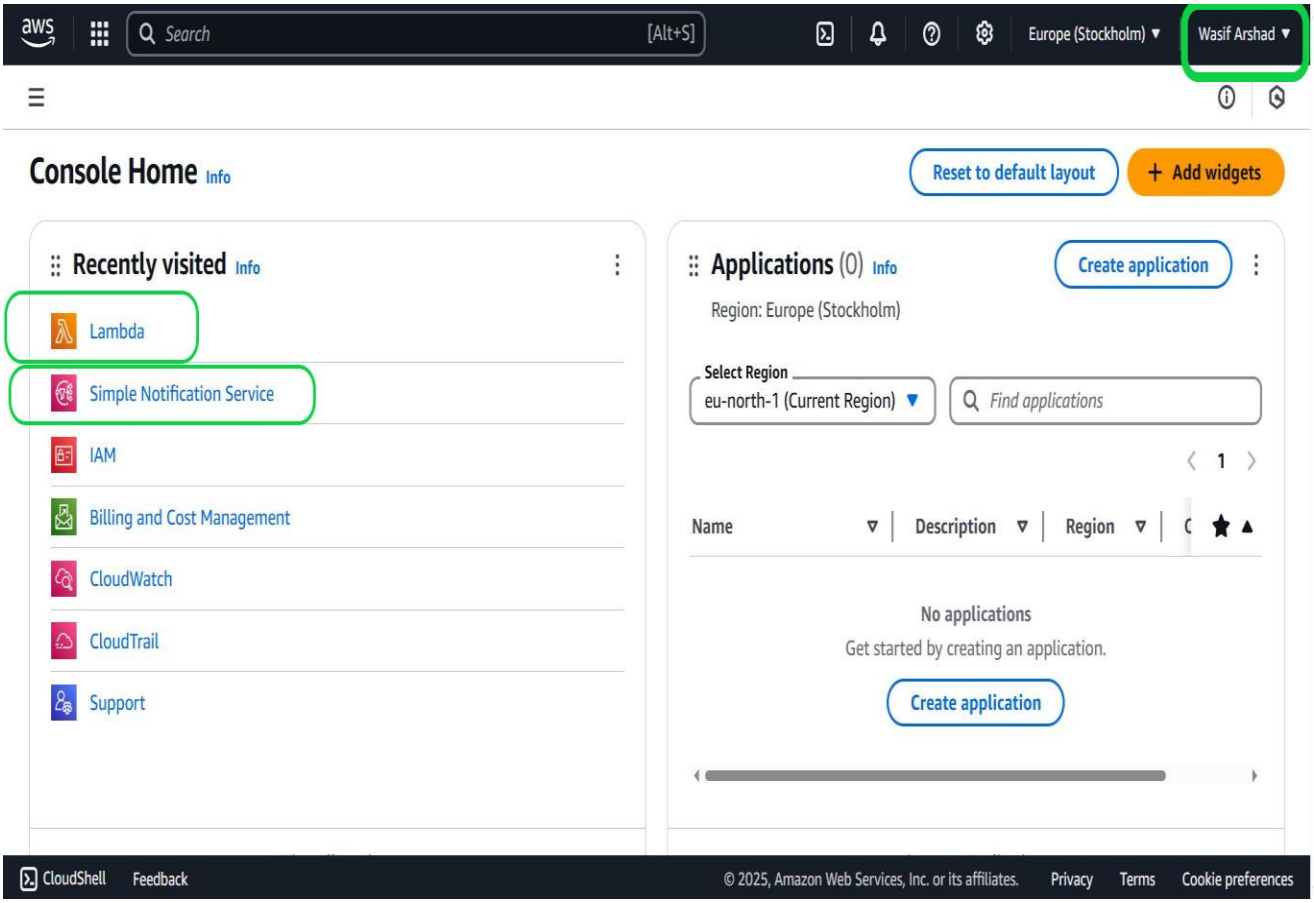


Figure 4.22: AWS Console Showing Recently Used Services

The simulation began by accessing the AWS Management Console and creating a custom AWS Lambda function. This function was designed to ingest glucose values and continuously evaluate them against predefined clinical thresholds. For this experiment, the upper and lower thresholds were set at 180 mg/dL and 80 mg/dL, respectively, which represent potential hyperglycemic and hypoglycemic states.

4.7.1 Accessing AWS Console and Setting Up Lambda Function

Inside the lambda environment, a Python script is used to generate the alert when a certain value cross a pre defined threshold value .

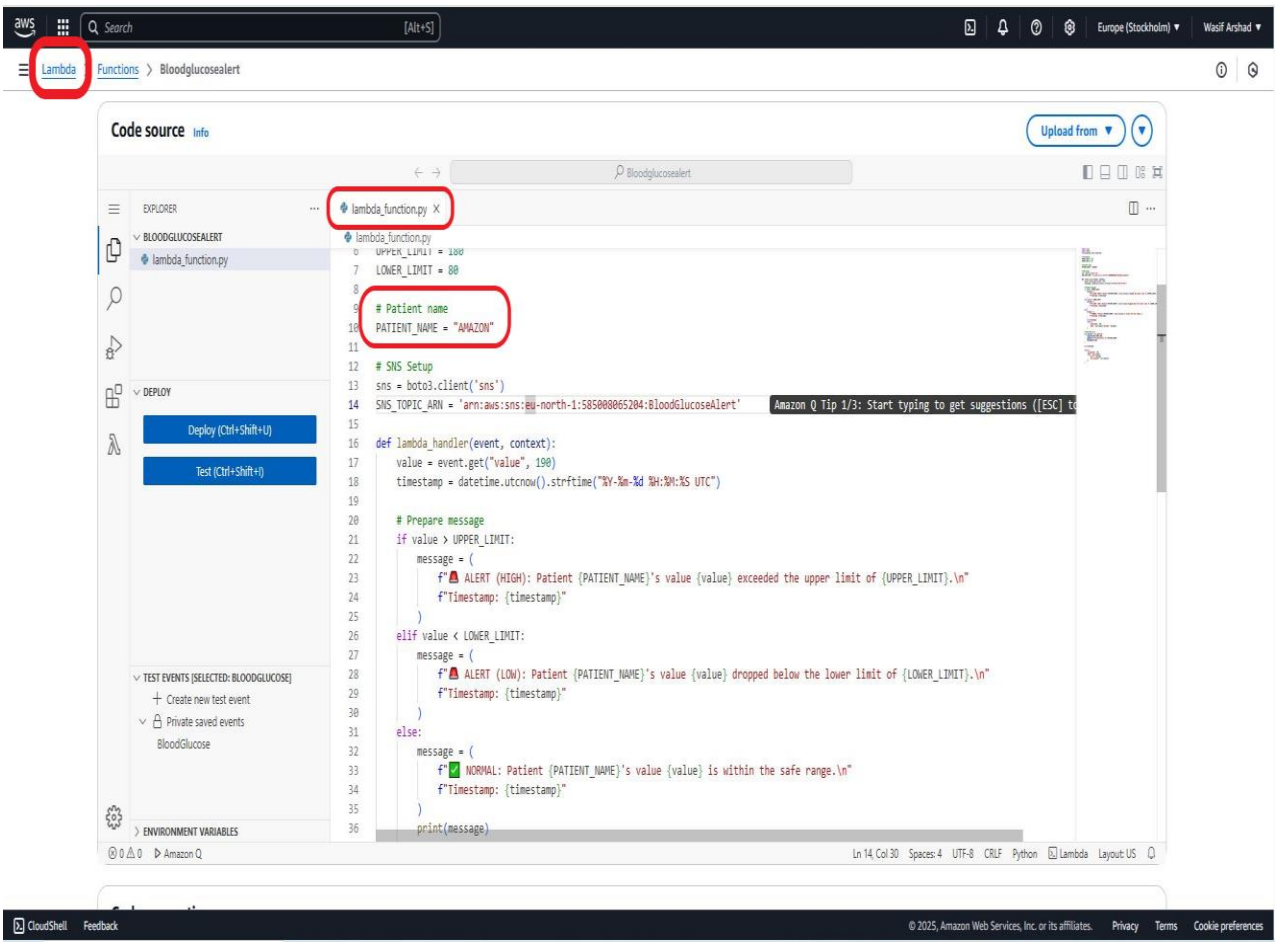


Figure 4.23: Trigger Alert Script On AWS Lambda Function

4.7.2 Configuring AWS SNS for Alert Generation

To ensure that on the basis of triggered function the real time alerts were generated and transmitted to the concerned persons and SNS topic is created and configured so that the SMS or Email alerts sent and received in time.

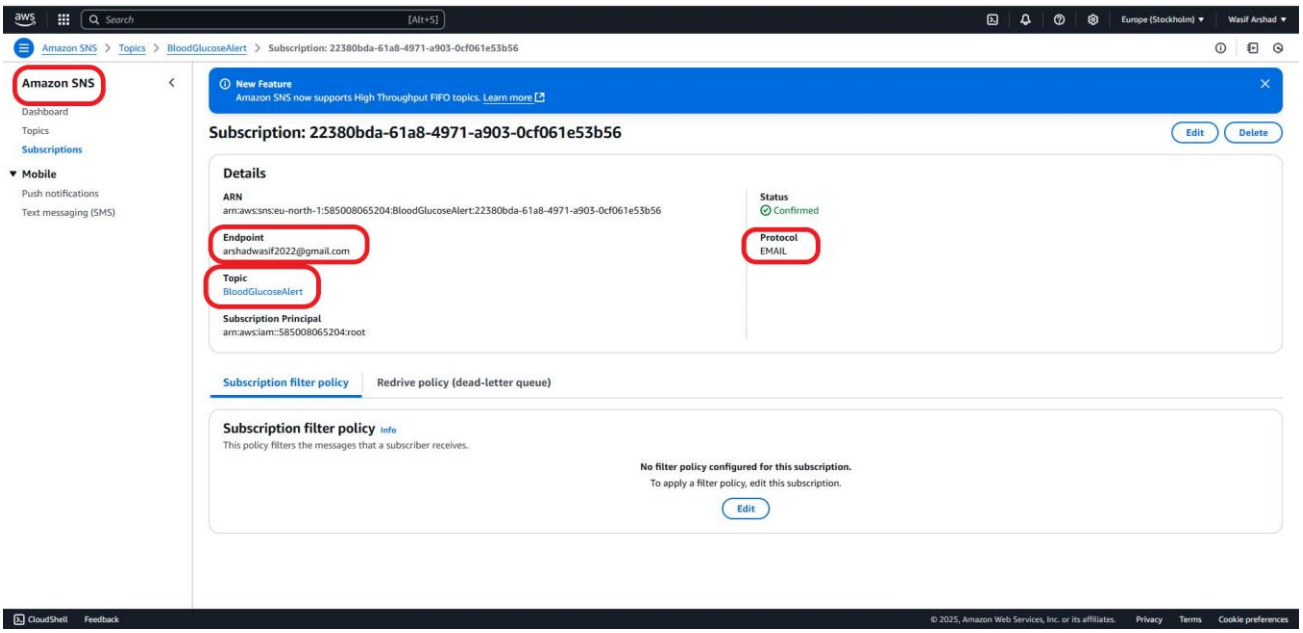


Figure 4.24: SNS Configuration On AWS SNS Page

4.7.3 Triggering Alerts Based on Static Threshold Conditions

In the final phase of the simulation, the Lambda function was tested with static values that represent abnormal glucose levels. For instance, test inputs such as 190 mg/dL (hyperglycemia) and 75 mg/dL (hypoglycemia) were provided. Each time these values breached the thresholds, the Lambda function successfully triggered SNS, and an automated email was sent to the registered recipient

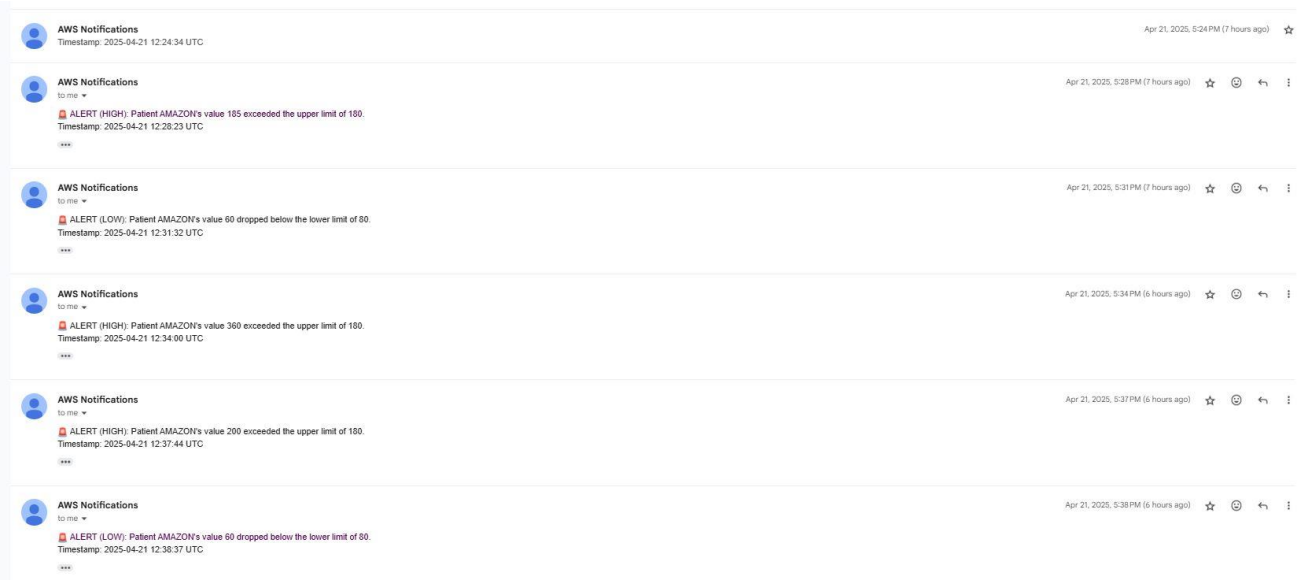


Figure 4.25: Received Email Log On Configured ID

In this research the static values of inputs or thresholds are used. Future implementations could feed live or forecasted SARIMAX outputs into the Lambda function via periodic invocations or event-based triggers, automating the full predictive-to-alert pipeline.

This simulation validated that the AWS-based alerting system operates reliably, is scalable, and is suitable for extension to more complex workflows, including those driven by machine learning or time-series predictions. Moreover, it emphasizes the utility of cloud-native services in building responsive and secure healthcare monitoring applications.

4.7.4 Data Upload to AWS Storage Using S3 Service

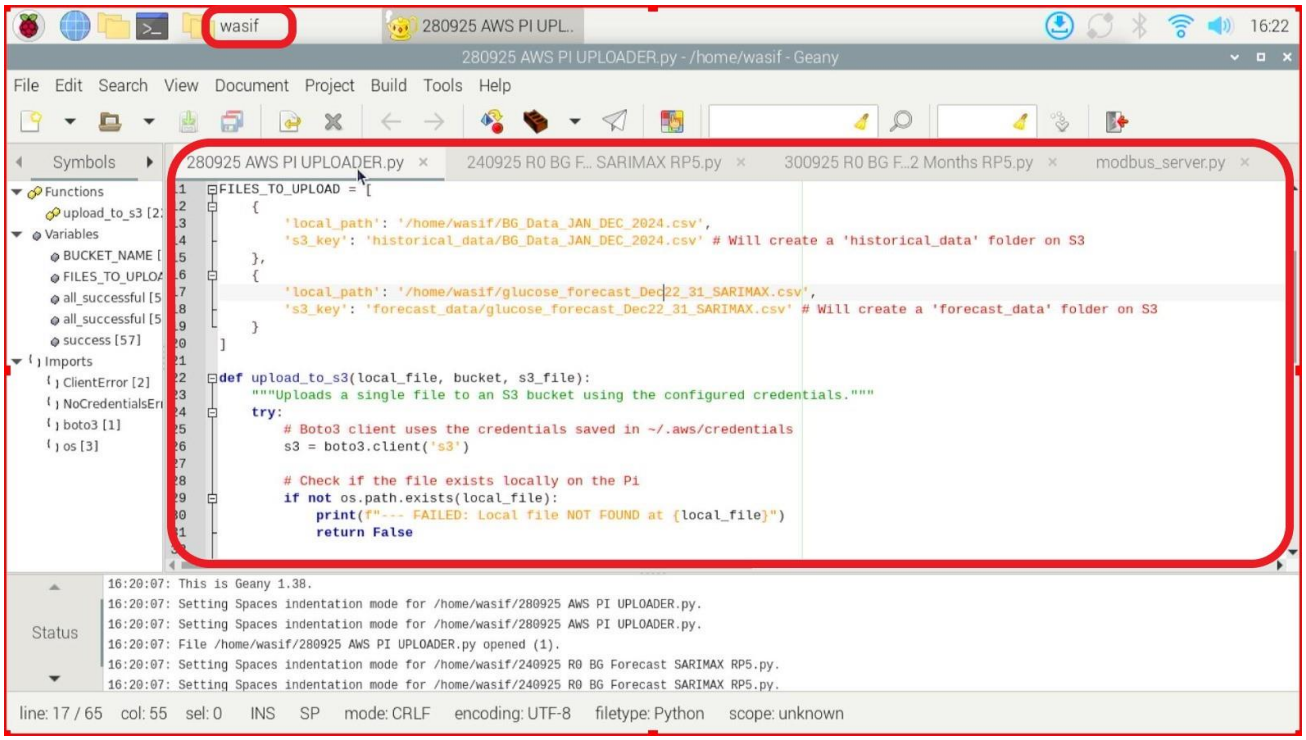


Figure 4.26: Execution of Python Script to Upload Data to AWS

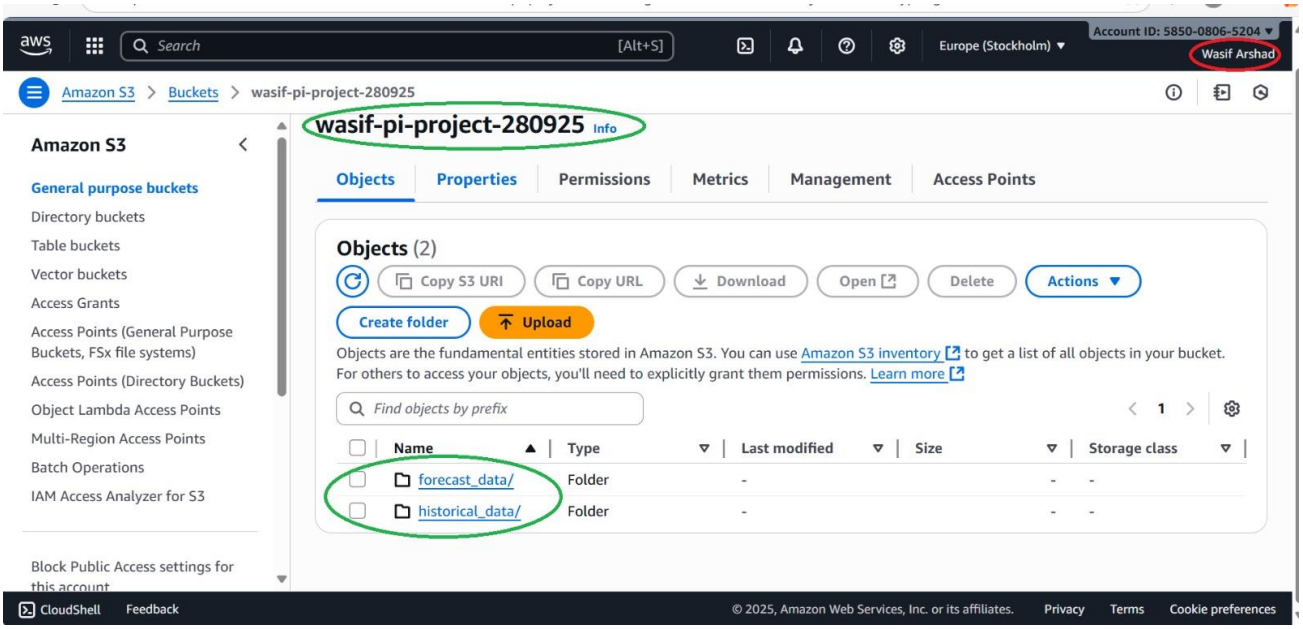


Figure 4.27: Uploaded Data Sets at AWS S3 Bucket

The snapshots used in the paper present the most crucial stage of the project, which is the process of data-transfer, and thus, illustrates how the localized Raspberry (PI) edge device and the Amazon Web Services (AWS) cloud storage integrate well. The former image shows the Python code 280925 AWS PI UPLOADER.py which is run in an integrated development environment on the Pi and makes sure that the code is set to use the boto3 client to upload major project artifacts. Namely, the script not only copies previous historical blood-glucose data into the S3 folder under historical data to target the historical-data folder, but also dry-copies the latest predictive data to undergo forecast data. This is done to justify the architectural design assertion presented in the thesis: the advanced SARIMAX prediction can be run effectively on the low-resource PI but the output is also being sent to the cloud to allow its remote access, long-term storage and further processing by final users or healthcare practitioners.

The second-screen shot shows the AWS S3 Console and provides a conclusive graphical evidence that the cloud-integration system is in effect. It shows the successful access to an S3 bucket wasif-pi-project-280925 that has the two major organizational directories forecasted and historical. In turn, it can be summarized that the Python script managed to run the upload commands.

- **Python Script**

Please refer to code **"A7-Data Uploader from Microcontroller to AWS S3 Bucket"** in APPENDIX A

4.8 Dash Board & Graphics

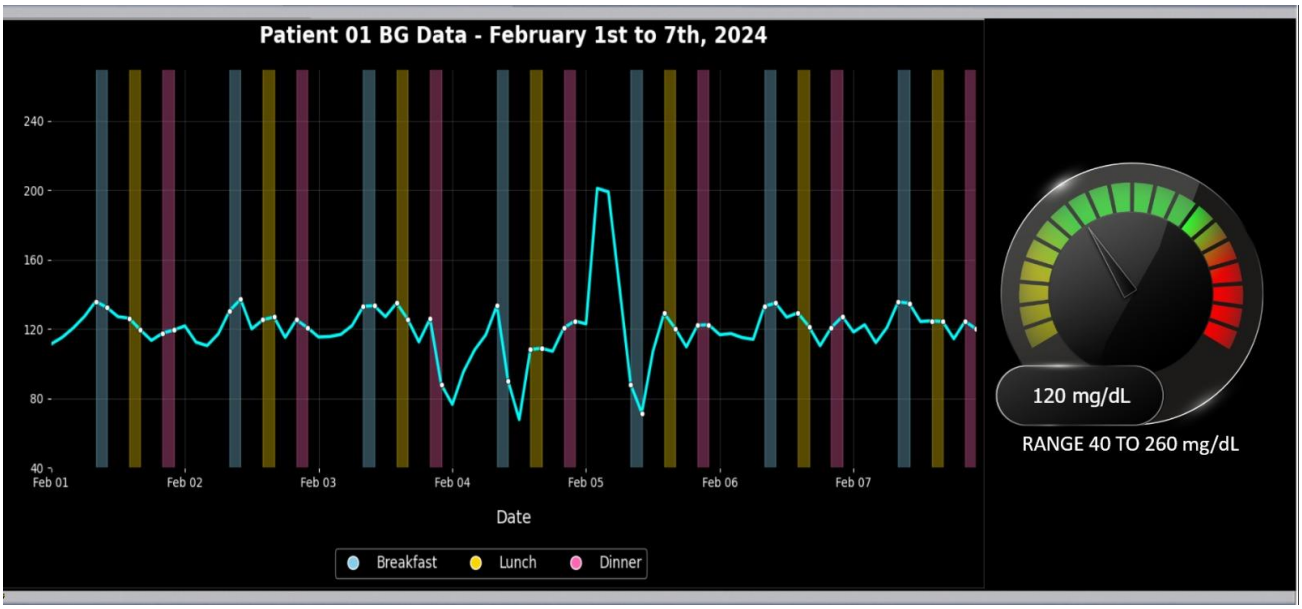


Figure 4.28: Graphics User Interface & Dashboard Design

Using RC studio, the creation of a real-time blood glucose (BG) monitoring dashboard was done. The aim was to come up with an interactive and responsive interface that will be able to visualize the trends in patient glucose along with meal timing and pre-established glycemic thresholds. The dashboard showed a running trend line on the BG and meal event (breakfast, lunch, dinner) indicators as well as a graphical gauge with a dynamic color scale, which indicated glucose levels as normal (green), or critical (red). Once the configuration scripts were complete, the dashboard was built and deployed to the MPWC (Mach Pro Web Controller) where it hosted the visualization interface and satisfied the role of exchanging data with other connected devices. Channels of communication and the synchronization settings were configured in the MPWC controller's configurations panel, which allowed glucose data and visualization parameters to be updated in real-time through its web interface, without having to use a separate host computer.

Raspberry 5 was set up in the deployed system; it was the main server that received the relationship of the MODBUS TCP/IP slave device (data acquisition and edge-level processing) (slave). Rather than making use of physical sensors, the Raspberry Pi obtained artificial readings of the glucose, used

preprocessing algorithms to reduce the noise effect and ensure integrity and sent the processed results to the MPWC controller at a regular rate. This data was also fetched by the controller and was automatically recalculated into the dashboard items essentially the trend plots, gauge indicators, and meal-time markers. This arrangement was able to achieve smooth USE of the MODBUS between the Raspberry 3 and the web controller to confirm the capability of the dashboard to serve as an interactive remote visualization system of real time glucose monitoring in the context of an IoT-enabled healthcare system.

- **Python Script**

Please refer to code **"A8- Configuring Raspberry PI 5 as MODBUS Server Device Over TCP/IP Network"** in APPENDIX A

4.9 Simulation of AES Encryption for Data Security

Data privacy and integrity is a critical issue in remote patient monitoring (RPM) systems. Since highly sensitive health metrics e.g. heart rate and blood glucose levels are relayed with edge devices and cloud servers, they fall victims to a range of cyber threats, such as interception, tampering, and unauthorized access. To address these risks, this section describes a simulation of the Advanced Encryption Standard (AES), a complex cryptographic algorithm that is symmetric-based, and implemented in a healthcare Internet of Things to prevent them.

The simulation will mimic the real-life situations associated with operation whereby patient information is coded before being sent into the network. Using the synthetically generated health data, we had applied an encryption-decryption loop and tested it thoroughly to determine both the functional feasibility and the physical security benefits that AES would offer in RPM systems.

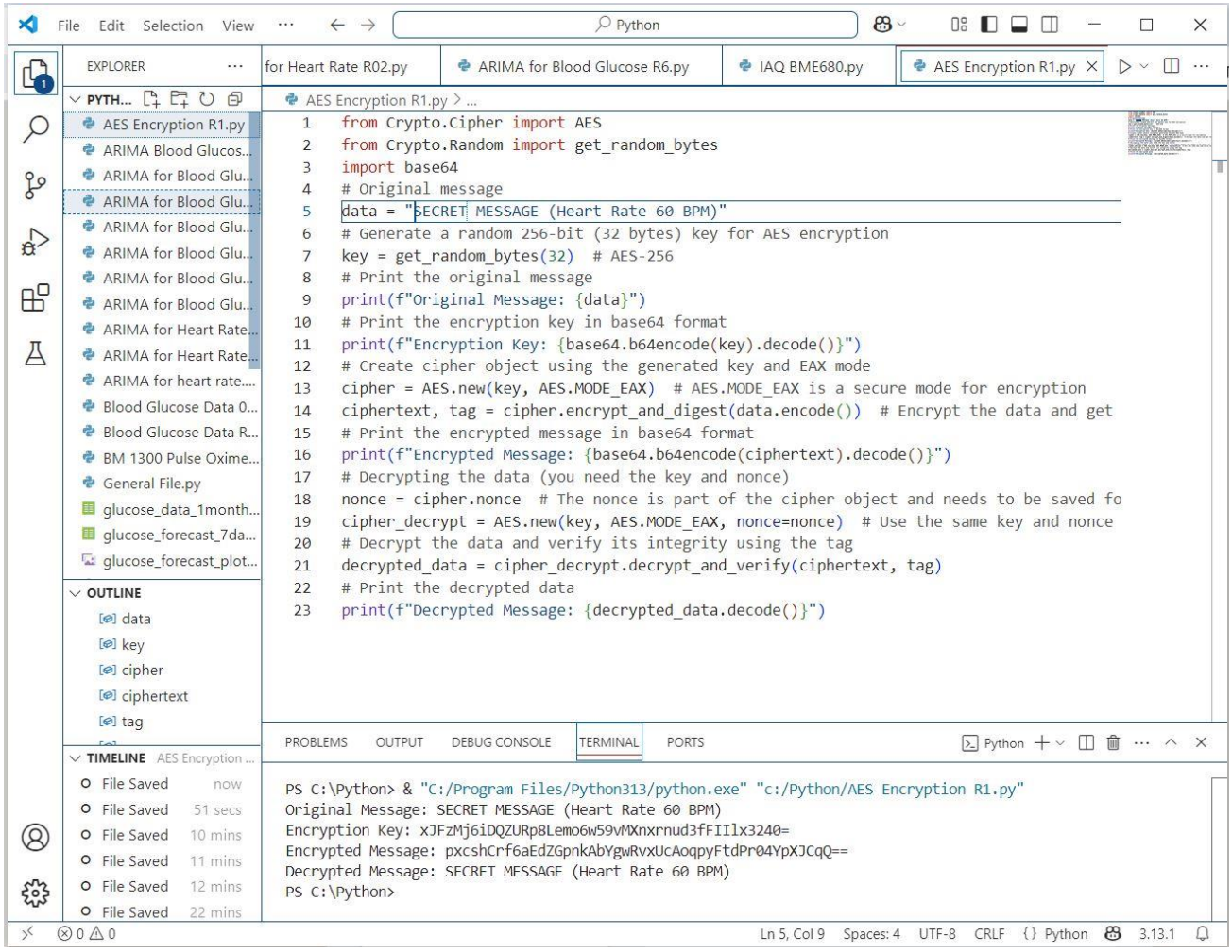


Figure 4.29: Simulation of AES Encryption using Python in VS Code

Figure 4.29 A python simulation, depicted in Figure 4.29, was built in Visual Studio Code (VS Code), which offers a friendly work environment on which to write the scripts of cryptography. Some of the modules that are imported to support safe encryption, generation of keys, and encoding process include Crypto Cipher AES, Crypto Random, and base64.

The data used in the simulation named SECRET MESSAGE (Heart Rate 60 BPM) shows sensitive health data that are typical of the Remote Patient Monitoring (RPM) data sets. The encryption process includes the following major steps:

- **Key Generation:** A 256-bit (32-byte) AES key was created by passing a cryptographically secure random number generator. Base64-Encoding was then done which allowed the key to be easily viewed and transmitted visually.
- **Cipher Initialization:** AES was set in EAX mode which offered confidentiality, as well as, authentication (integrity) at a reasonable cost. Pattern-based attack mitigation EAX is widely considered to be better than Electronic Codebook (ECB) mode.
- **Encryption and Encoding:** The plaintext message was encrypted using AES cipher and the resulting cipher text was encoded using Base64 which also simulates the format that could be safely transmitted.
- **Decryption and Verification:** Cipher text was decrypted and deciphered with the same cipher key and cipher mode. It was known that the original plaintext could be correctly recreated to ensure that the implementation was correct.

This simulator proves that the utilization of AES encryption can be utilized trustworthily to protect healthcare information upon communication between Internet-of-Things equipment (e.g., Raspberry Pi) and cloud providers (e.g., Amazon Web Services). The findings highlight the fact that security is an inseparable aspect of RPM systems, not something regarding which it should be considered after the fact.

The terminal output confirms:

- The original plaintext message
- The Base64-encoded AES key
- The encrypted (cipher text) output
- The successfully decrypted plaintext, which matches the original

Such encryption ensures that even if data packets are intercepted during transmission, the actual patient information remains inaccessible without the correct decryption key.

4.10 Performance Evaluation and Benchmarking

To confirm the feasibility of the suggested hybrid edge -cloud remote patient monitoring (RPM) system, we used a performance evaluation that focused on the efficiency of execution, computational

load, and system responsiveness. The simulation experiments considered the most critical elements such as a SARIMAX-based forecasting deployed to the edge (Raspberry 5) and the automated alerting process mediated by AWS Lambda and SNS. Regarding load of the system, CPU and RAM usage on the Raspberry Pi were reasonable when running the SARIMAX, which confirms the fact that lightweight forecasting is achievable without depending on cloud computing capabilities. Additionally, the communication channel between the Raspberry and AWS services, particularly, the alerts through the SNS was put through simulations that considered the realistic conditions of the network. The results reveal that the hybrid system can be scaled and, in the future, can support an increasing number of sensors or patients assuming that the inherent resource limitations of edge devices are considered in future deployments.

4.10.1 Forecasting Performance on Edge Device (Raspberry Pi 5)

SARIMAX model was implemented locally in a Raspberry Pi 5 to replicate real-time time- series prediction in the edge layer. It was done to determine whether lightweight statistical prediction algorithms would run within the resource constraints of a low-power device.

- Execution Time: The SARIMAX model is capable of generating 7-day heart-rate and glucose predictions in less than a dozen seconds and has a low computing efficiency level which coincides with the near-real-time response that the model supports.

- CPU & Memory Utilization: At peak performance, the CPU use was briefly peaked at around sixty percent, with the RAM usage at less than five hundred megabytes, which means that the device can be used to run forecast tasks without performance loss or thermal impaction.
- Feasibility: The findings indicate that even predictive analytics with moderate complexity (SARIMAX) can be used on embedded platforms, which allows proactive health information to be presented on local embeds instead of relying on cloud computing (immediately).

4.10.2 Edge Cloud Communication Performance:

Simulated network environments were used to evaluate the resilience and efficiency of data exchange between the Raspberry Pi and AWS cloud services:

- **Data Uploads and Alert Triggers:** The latency for sending encrypted forecast results and triggering alerts remained under 3 seconds under moderate simulated network congestion.

- **Packet Loss Tolerance**: The system handled minor network delays (≤300ms) and packet loss (≤1%) without disrupting forecast uploads or alert generation, confirming operational reliability.

4.10.3 System Scalability and Load Considerations:

The modular design of the hybrid architecture allows horizontal scaling, enabling the addition of multiple patient devices without drastically increasing system load.

- **Multi-Patient Forecast Simulation:** Testing with simulated data streams for five patients revealed that the Raspberry Pi could concurrently execute SARIMAX forecasting for multiple datasets with sequential scheduling, provided memory usage was monitored.
- **Bottlenecks**: The primary limiting factor remains the edge device's RAM capacity, which may constrain simultaneous real-time operations if the number of monitored parameters or patients increases substantially. Solutions include offloading heavier tasks to cloud services or upgrading hardware.

4.11 Comparative Analysis

In this particular work, the hybrid edge-cloud architecture was compared to two traditional setups cloud and local architectures one is completely dependent on local processing while the other one is completely dependent on cloud to for data storage, processing and alerts generation

Cloud dependent system contains high scalable processing capacity and storage, which makes it feasible for large datasets handling, as well performing complex analysis. However, the systems require a reliable internet connectivity throughout the time, which is not usually possible in almost every part of the region. Another issue is latency which involves the transfer of data in and out of the cloud to process it and hence slowing down time-driven alerts and responses. Even though the performance of cloud platforms can be scaled effectively and can be used to trigger notifications in real-time through the built-in services like AWS SNS, they may suffer a breakdown during network effects, thus compromising performance when the activity is critically required in healthcare settings.

Local based systems on the other hand have better response times due to high speed data processing and negligible latency. These systems make them less reliant on networking to perform forecasting operations because they execute them directly on edge devices like the Raspberry Pi. The scalability and combination with the cloud-based alerting systems such as AWS SNS, however, are limited by the limited computational resources available in edge hardware. In turn, it hinders the spread of alerts outside of the local setting and hampers the inability to handle data centrally or store data over a period of time.

The hybrid system edge-cloud which integrates the benefits of both of the models SARIMAX forecasting real-time running on the Raspberry Pi makes sure that the latency is minimal, and the system can run continuously even when the internet connectivity is not available . This structure enables it to manage workload distribution as well as have an offloading of the heavy workload alerting logic to the cloud without affecting the responsiveness of the local decision making

4.12 Challenges Faced During Simulation

There are various challenges that arose during the simulation stage of the hybrid system. It was one of the major challenges related to real-time integration of forecasting logic and AWS lambda. The fact that Lambda triggers are asynchronous and the lack of a continuous data flow that stems out of the forecasting module did not allow a direct connection between the SARIMA output and the Lambda

function. As a result, a workaround was created, which consists of the simulation of the threshold-based alerts with the constant values. This was also effectively used in a demonstration mode although this did not completely recreate a live, prediction driven scenario of an alert.

The other problem was the simulation of sensor behavior based on mock data. Similar to the fabricated values allowing the testing of the forecasting and alerting workflows, the fabricated values could not reflect the unpredictability and variability of the real physiological signals. Also, the IAM roles and permissions of the AWS services, in particular, the Lambda-to-SNS interactions, had to be carefully configured, or it may result in the security and functionality breaches. Such setbacks and technical challenges that are expected of a prototype stage, however, provide valuable information towards some improvements in the system in the ensuing physical implementation phase.

AWS Lambda and SNS were utilized in order to simulate responsive healthcare alerts. Despite the fact that the Lambda operation did not rely on the output of the SARIMAX because of the concepts of simulations, it was able to show a real-time notification upon threshold violations. This style of decoupling proved the ability of the system to support asynchronous processing which is an essential key attribute of scalable healthcare applications.

The aspect of data security was taken care of by simulating the AES encryption in python. The implementation had ensured that encrypted transmission of patient sensitive health information using a symmetric-key algorithm was possible thus improving privacy to patients and integrity of the system.

The efficiency in performance testing enabled by the system to handle forecasting operations on a regional scale without going beyond the computational capabilities of the Raspberry Pi. The overhead of cloud communication was not too high, and the hybrid method has provided a decent allocation of work. The comparative analysis proved that, the hybrid architecture was more efficient than both the cloud-only model and the local-only model in minimization of the latency, and minimization of network dependency and the ability to disperse alerts reliably.

Although the simulation was full of strong validation of the core functionality of the system, there were several issues that were faced, namely, real-time integration of forecasting and cloud-based alert features, use of mock data, and configuration of AWS services. However, these shortcomings also provided valuable information in terms of system improvement.

The lessons learned helped in highlighting the need to have a modular architecture and the separation of forecasting and alert logic, and the use of realistic mock data. Under the physical implementation, it is planned to introduce real sensor integration, improved human-machine interface (HMI) and dynamic alerting. Other features like SMS or mobile app notifications would also be expected to enhance usability among end-users.

CHAPTER 5
Conclusion & Future Work

5.1 Overview

This chapter concludes with a general conclusion and presents future directions of the proposed hybrid edge-cloud healthcare monitoring system based on the IoT. It provides a brief summary of the main aims, used techniques and results achieved throughout the research. The key objective of the study was to develop, deploy, and evaluate an integrated remote patient monitoring (RPM) model that unites real-time data collection, prediction, and auto alarms by implementing edge computing (Raspberry Pi 5) in combination with cloud computing (AWS). The chapter highlights the most important findings based on the model implementation, system performance and simulation findings. In addition, it outlines the boundaries experienced in the course of the inquiry and suggests the future path of improving the system and its implementation in the intelligent monitoring procedure of healthcare.

5.2 Summary of Work

The current research is related to the creation of a hybrid edge-cloud design that can be used to provide sustained health monitoring and predict essential values. The system was designed to be operated by Raspberry Pi 5 as an edge node, and it was connected with cloud-based applications (AWS Lambda and SNS) to generate automated alerts. The edge tier was assigned with the task of acquiring data, pre-processing it, encrypting it, and making a local prediction, and the cloud tier provided the capability of scaling, long-term storage, and notifying.

To measure the performance, synthesized datasets containing data on the blood glucose levels and heart -rate performance was used to test and validate the predictive models. SARIMAX and Bidirectional LSTM models have been applied to forecast the glucose level at various time scales. When comparing the results of both the SARIMAX and the LSTM model, it was observed that it was more efficient in smaller training data sets, yet the LSTM model exhibited a high learning ability in the case of larger training data sets. There was also the collaboration of the AWS which practically automated warnings about abnormal measurements, which also supported the feasibility of real-time monitoring of patients in a hybrid setting.

Together, the system achieved the reliability of forecast quality and effective cloud-edge interconnectedness, which establishes its appropriateness to practice in less urban health care settings where timely response and the safety of data are essential.

5.3 Major Findings

The experimental test produced a number of vital results that highlight the performance and effectiveness of the system. The SARIMAX model was found to be highly accurate and stable in a short-term forecasting, initially when it has a small volume of data, resulting in small values of mean absolute error (MAE) and root-mean-square error (RMSE); but its performance declines with increasing data complexity. On the other hand, the Bidirectional LSTM model showed gradual increases in the performance with larger datasets, and it was found to effectively learn the complex patterns of glucose fluctuations and when it attains minimal errors in forecasting.

Results also proved that the accuracy of the model increases significantly with the amount and diversity of data as the error indicators steadily decrease with the growth of the training time. In addition, the AWS Lambda-SNS integration could be described as a reliable cloud-based alerting system, which created and disseminated notifications regarding irregular readings automatically, without human control.

Finally, the research supported the feasibility of the Hybrid Edge-Cloud Healthcare system being implemented using inexpensive hardware, like the Raspberry Pi 5. The system was also able to process, encrypt, and visualize the local data and maintain the secure communication with the cloud services, which makes it a strong and cost-effective solution to the problem of monitoring the patients remotely.

5.4 Limitations

In spite of the achievements of the implemented system and the positive outcomes, there are a few limitations of the proposed system that cannot be ignored. To begin with, the data employed was simulated as opposed to actual continuous glucose monitoring (CGM) sensors. Although this can be experimented upon, this method might not be suitable to the variability and noise found in physiological data.

Second, the SARIMAX and LSTM networks were trained and tested during a small time period and limit them to generalization over extended periods or with different patient profiles. LSTM in

whatsoever applies to bigger datasets to perform optimally, and its training time and computing expense increase correspondingly.

Third, edge to cloud integration used a reliable network connection to synchronize live between the Raspberry Shanghai and the AWS services. Data transmission and the creation of alerts can be impacted by any disruption at the network. Also, it was not tested in a real-life situation of data encryption security on multi-user data access or on the most common environment of cybersecurity threats (although AES encryption was simulated to support data security).

Finally, the system did not emphasize the dynamic combination of other important parameters to define a final model, like heart rate, blood pressure, and oxygen saturation, which can restrict its usage as a holistic patient remote monitoring (RPM).

5.5 Conclusion

The paper developed and tested a hybrid edge–cloud-based healthcare monitoring system capable of predicting continuous blood glucose levels and raising an alarm. By deploying Raspberry 5 as the edge node and AWS cloud services to process the data on the backside, the system proved that the system can be quite effective in terms of the real-time data processing, model execution, and automated notifications. The SARIMA and Bidirectional LSTM forecasting models were applied to model and predict CGM glucose levels by using simulated CGM data. The relative analysis showed that the LSTM predictive, however, performed better, because of its capability to learn complex nonlinear trends, whereas the SARIMAX model offered better interpretability and strength of performance on smaller datasets.

The system has effectively confirmed the suitability of edge-based analytics in conjunction with the use of the cloud to support the logic-based remote health monitoring. It was able to effectively manage on-site processing, reduced the time lag on data and ensured safe cloud replication using encryptions and services offered by AWS. This architecture can be used to build real time patient management system, especially the diabetic patients who need to be monitored on their glucose levels and detected anomalies in time.

The findings of this study reiterate the fact that the combination of edge computing and cloud analytics might be used to enhance the scalability, responsiveness, and reliability of healthcare monitoring systems. The need to use both SARIMAX and LSTM models makes the framework a common balance

between interpretability and predictability and the potential solution to personalized glucose monitoring. Even though the simulated data was used in the study, the architecture and the methodology can be easily applied to real-world continuous glucose monitoring (CGM) devices. Properly calibrated, this system may be expanded to log other critical measurements like heart rate, blood pressure and oxygen saturation to make it an intelligent remote healthcare environment.

5.6 Future Work & Improvements

The present study was able to create an artificialized hybrid edge cloud healthcare monitoring with the help of SARIMAX and LSTM prediction mechanisms. This will encompass connecting glucose and heart rate sensors to the Raspberry Pi 5 to allow the gadget to automatically read and interpret as well as can relay live patient data into the cloud without any human intervention.

The automation of data management processes will also be implemented into future release of the system as the Raspberry and the means to revise the data outdated every once in a while, and upload only the most recent information will give the system a chance to use the entire storage capacity and use the bandwidth more effectively.

Further advances might comprise the expansion of forecasting possibilities of the system to multi-parameter health information, the use of federated learning in the creation of privacy-safe model training, the creation of the interface with the mobile application that allows patient feedback and remote control.

5.7 Main Contributions of Study:

Development of a hybrid edge–cloud RPM system capable of real-time glucose and heart rate monitoring.

- Implementation of SARIMAX and LSTM models for accurate short-term forecasting.
- Integration of automated alert generation and secure cloud-based data storage.
- Demonstration of system resilience, low-latency edge processing, and efficient bandwidth usage.

REFERENCES

[1] N. M. Ghazaly and N. Jain, "IoT-Based Health Monitoring System: Design, Implementation, and Performance Evaluation," Research Journal of Computer Systems and Engineering, vol. 4, no. 2, pp. 144–159, 2023. https://technicaljournals.org/RJCSE/index.php/journal/article/view/84technicaljournals.org

[2] M. Kumar, "Optimizing Security for Remote Patient Monitoring with Edge Computing Strategies," International Research Journal on Advanced Engineering and Management (IRJAEM), vol. 2, no. 5, May 2024 https://goldncloudpublications.com/index.php/irjaem/article/view/255Goldn Cloud Publications

[3] K. S. Kumar and D. Eshwar, "Optimizing IoT Healthcare Security with Advanced Hybrid Encryption and Data Embedding Techniques," *History of Medicine*, vol. 10, no. 1, Feb. 2024. https://historymedjournal.com/HOM/index.php/medicine/article/view/717

[4] H. Aissaoua et al., "Integrating Homomorphic Encryption in IoT Healthcare Blockchain Systems," *Information Systems and Informatics*, vol. 29, no. 5, pp. 1–10, Oct. 2024. https://www.iieta.org/journals/isi/paper/10.18280/isi.290501

[5] A. Alif, K. F. Hasan, J. Laeuchli, and M. J. M. Chowdhury, "Quantum Threat in Healthcare IoT: Challenges and Mitigation Strategies," arXiv preprint arXiv:2412.05904, Dec. 2024. https://arxiv.org/abs/2412.05904

[6] How IoT Is Transforming Healthcare Cybersecurity in 2025, Toxigon Blog, Jan. 2025: https://toxigon.com/the-impact-of-iot-on-healthcare-cybersecurity

[7] Healthcare IoT Security: Protecting Patients in 2025, Toxigon Blog, Jan. 2025. https://toxigon.com/healthcare-iot-security

[8] The US proposes rules to make healthcare data more secure, The Verge, Dec. 2024. https://www.theverge.com/2024/12/28/24330878/the-us-proposes-rules-to-make-healthcare-data-more-secure

[9] A. Abebe, A. Y. Zewdie and A. Tegegne, "IoT Based Remote Patient Monitoring System," Journal of Healthcare Engineering, vol. 2022, pp. 1–12, 2022. https://www.hindawi.com/journals/jhe/2022/5064132/

[10] M. Hussain, S. Khan, and R. Afzal, "Time Series Analysis and Forecasting for Healthcare Using ARIMA and SARIMAX Models," *Procedia Computer Science*, vol. 198, pp. 652–658, Feb. 2022.:https://www.sciencedirect.com/science/article/pii/S1877050922001391

[11] R. Djemame, D. Armstrong, M. Papazoglou, and K. Kritikos, "A Framework for Automating Cloud Service Management," *Future Generation Computer Systems*, vol. 95, pp. 597–612, Feb. 2019.https://www.sciencedirect.com/science/article/pii/S0167739X18321624

[12] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 8th ed., Pearson, Jul. 2022.https://www.pearson.com/en-us/subject-catalog/p/cryptography-and-network-security-principles-and-practice/P200000003826

[13] N. A. Kassab, R. Z. Homod, and R. T. Khalaf, "Design of a Smart Healthcare Monitoring System Using IoT and Raspberry Pi," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 29, no. 1, pp. 156–163, Jan. 2023**.** https://ijeecs.iaescore.com/index.php/IJEECS/article/view/30767

[14] M. Sethi and S. Anand, "Secure Communication in IoT Using Advanced Encryption Standard (AES)," *International Journal of Advanced Research in Computer Science*, vol. 12, no. 6, pp. 43–47, Jun. 2021**.** https://ijarcs.info/index.php/Ijarcs/article/view/6758

[15] Iqra Batool, "Real-Time Health Monitoring Using 5G Networks: A Deep Learning-Based Architecture for Remote Patient Care," *arXiv preprint arXiv:2501.01027*, Jan. 2025. https://arxiv.org/abs/2501.01027

[16] Binariks, "The Role of Edge Devices in Hybrid Healthcare Monitoring," https://binariks.com/blog/edge-computing-for-healthcare-data

[17] Advancements in Data Security for IoT-Based Healthcare Systems," *Journal of Cloud Computing*, 2024. https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-024-00654-4

[18] Developing Fail-Safe Mechanisms in Remote Patient Monitoring Systems," *arXiv preprint*, arXiv:2301.03551, 2023 https://arxiv.org/abs/2301.03551

[19] Hybrid Models for Scalable Healthcare Delivery," *Journal of Cloud Computing*, 2024. https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-024-00654-4

[20] P. Sharma, "The Evolution And Future Of Remote Patient Monitoring," *Forbes*, Jun. 2, 2022. https://www.forbes.com/councils/forbestechcouncil/2022/06/02/the-evolution-and-future-of-remote-patient-monitoring/

[21] The Many Benefits of RPM for Chronic Care Management, *Medek RPM*, Sep. 19, 2023 https://www.medekrpm.com/blog/Many-Benefits-of-RPM-for-Chronic-Care-Management

[23] The Benefits of RPM for Chronic Disease Management, *ProactiveWellness*, Apr. 10, 2023. https://prowellcare.com/rpm-for-chronic-disease-management/

[24] Chronic Care Management for Medicare, *Verywell Health*, 2021. https://www.verywellhealth.com/medicare-chronic-care-management-5199917

[25] Impact of remote patient monitoring on clinical outcomes: an updated meta-analysis of randomized controlled trials," *PMC*, 2019. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6550143/

[26] S. Shaik et al., "Remote patient monitoring using artificial intelligence: Current state, applications, and challenges," *WIREs Data Mining and Knowledge Discovery*, vol. 13, no. 1, 2023 https://wires.onlinelibrary.wiley.com/doi/10.1002/widm.1485

[27] Benefits and Challenges of Remote Patient Monitoring as Perceived by Health Care Practitioners: A Systematic Review," *PMC*, 2023. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10730976/

[28] AI for Remote Patient Monitoring: A Lifeline for Rural Healthcare, *IIInigence*, 2023 https://www.iiinigence.com/post/ai-remote-patient-monitoring-rural-healthcare

[29] A systematic review of the impacts of remote patient monitoring (RPM) interventions on safety, adherence, quality-of-life and cost-related outcomes, *PMC*, 2023. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC11258279/

[30] R. Sharma et al., "Edge Computing in Healthcare: A Review on Current Trends, Challenges and Future Directions," *IEEE Access*, vol. 9, pp. 154409–154432, 2021. https://ieeexplore.ieee.org/document/9562639

[31] S. Islam et al., "A Review on Edge Computing in Healthcare: Architecture, Applications, and Challenges," *Future Generation Computer Systems*, vol. 114, pp. 255–266, 2021. https://doi.org/10.1016/j.future.2020.08.005

[32] A. M. Rahmani et al., "Exploiting Smart e-Health Gateways at the Edge of Healthcare Internet-of-Things: A Fog Computing Approach," *Future Generation Computer Systems*, vol. 78, pp. 641–658, 2018. https://doi.org/10.1016/j.future.2017.02.014

[33] N. H. Mahmood et al., "IoT Based Health Monitoring System Using Raspberry Pi: A Review," *Procedia Computer Science*, vol. 199, pp. 157–163, 2022. https://doi.org/10.1016/j.procs.2022.01.020

[34] R. Shah et al., "Implementation of Real-Time Health Monitoring System using Raspberry Pi," *IEEE International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, 2021.: https://ieeexplore.ieee.org/document/9509944

[35] AWS for Healthcare – Remote Patient Monitoring, *Amazon Web Services*, 2023. https://aws.amazon.com/health/solutions/remote-patient-monitoring/

[36] HIPAA Eligible Services on AWS, *AWS Compliance*, 2023. https://aws.amazon.com/compliance/hipaa-eligible-services-reference/

[37] J. N. Al-Karaki and M. R. Ahmad, "Cloud-Based Remote Patient Monitoring Systems: A Survey," *Journal of Healthcare Engineering*, 2021.https://doi.org/10.1155/2021/9970936

[38] V. Sharma et al., "A Secure Hybrid Edge–Cloud Architecture for Health Monitoring Systems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, pp. 9311–9325, 2021. https://doi.org/10.1007/s12652-020-02534-4

[39] B. S. Chauhan et al., "A Comparative Study of Edge–Cloud and Fully Cloud Architectures in Healthcare Monitoring Systems," *Sensors*, vol. 21, no. 12, 2021.https://www.mdpi.com/1424-8220/21/12/3989

[40] R. H. Shaker and K. S. Al-Turjman, "Predictive modeling of vital signs using machine learning in healthcare," *IEEE Access*, vol. 9, pp. 41230–41245, Mar. 2021. https://ieeexplore.ieee.org/document/9374568

[41] A. Ahmed et al., "Early detection of health deterioration using time series forecasting in RPM," *Sensors*, vol. 21, no. 4, pp. 1273–1288, Feb. 2021. https://www.mdpi.com/1424-8220/21/4/1273

[42] M. Q. Raza et al., "Time series prediction of vital signs in remote patient monitoring systems," *Healthcare Analytics*, vol. 2, pp. 100023, Aug. 2022. https://www.sciencedirect.com/science/article/pii/S2772753X22000023

[43] S. Kumar and T. Garg, "Application of ARIMA in healthcare: A case study on heart rate monitoring," *Procedia Computer Science*, vol. 167, pp. 2223–2230, Dec. 2020. https://www.sciencedirect.com/science/article/pii/S1877050920314115

[44] H. Zhang et al., "Limitations of classical time series models for medical data," *Journal of Biomedical Informatics*, vol. 113, pp. 103620, Apr. 2021. https://www.sciencedirect.com/science/article/pii/S1532046421000832

[45] L. Gonzalez et al., “Forecasting patient vitals using SARIMA and SARIMAX models,” *IEEE Journal of Biomedical and Health Informatics*, vol. 25, no. 10, pp. 3872–3880, Oct. 2021. https://ieeexplore.ieee.org/document/9460097

[46] M. M. T. Ahmed, M. R. A. Kadir, and A. K. M. A. H. Bhuiyan, "Real-time forecasting of health vitals using SARIMAX for remote patient monitoring," *IEEE Access*, vol. 10, pp. 4317–4325, 2022. https://doi.org/10.1109/ACCESS.2022.3145879

[47] M. Q. Raza et al., “Time series prediction of vital signs in remote patient monitoring systems,” *Healthcare Analytics*, vol. 2, pp. 100023, Aug. 2022. https://www.sciencedirect.com/science/article/pii/S2772753X22000023

[48] A. Ali, T. F. Al-Somani, and M. N. Mohammed, “Temperature Prediction Using Transformer–LSTM Deep Learning Models and SARIMAX from a Signal Processing Perspective,” *Applied Sciences*, vol. 15, no. 17, p. 9372, 2025. https://doi.org/10.3390/app15179372

[49] R. R. Sharma, P. Kumar, and S. K. Mishra, "Real-time healthcare monitoring and alerting system using AWS SNS for patient care," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 8, no. 1, pp. 22-30, 2023. https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-023-00334-7

[50] A. Bouslama, Y. Laaziz, A. Tali, and M. Eddabbah, “AWS and IoT for real-time remote medical monitoring,” *International Journal of Intelligent Enterprise*, vol. 7, no. 1, pp. 1–15, Jan. 2021. https://www.researchgate.net/publication/335225229_AWS_and_IoT_for_real-time_remote_medical_monitoring

[51] R. Agarwal, A. Bhatnagar, and Y. Bindlish, “Visualize AWS Health events using Amazon Managed Grafana,” *AWS Cloud Operations Blog*, Dec. 2023**.** https://aws.amazon.com/blogs/mt/visualize-aws-health-events-using-amazon-managed-grafana/

[52] S. Hashemi, S. H. Islam, and M. S. Hossain, “Security and privacy in IoT-based healthcare: A survey,” *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4473–4494, Mar. 2021. https://ieeexplore.ieee.org/document/9216565

[53] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, “Middleware for Internet of Things: A survey,” *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, Feb. 2016. https://ieeexplore.ieee.org/document/7377674

[54] A. Mosenia and N. K. Jha, “A comprehensive study of security of Internet-of-Things,” *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 586–602, Oct.–Dec. 2017. https://ieeexplore.ieee.org/document/7488250

[55] R. Khan and K. Salah, “IoT security: Review, blockchain solutions, and open challenges,” *Future Generation Computer Systems*, vol. 82, pp. 395–411, May 2018. https://www.sciencedirect.com/science/article/pii/S0167739X17315765

[56] S. K. Dash, S. Mohapatra, and P. K. Pattnaik, “A survey on data security and privacy issues in IoT,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1802–1815, May 2022. https://www.sciencedirect.com/science/article/pii/S1319157818302647

[57] H. Alelaiwi, “Secure real-time health data aggregation in cloud-assisted IoT-based healthcare monitoring system,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 2, pp. 219–227, Feb. 2022. https://www.sciencedirect.com/science/article/pii/S1319157819302281

[58] B. Carter et al., "Hybrid IoT-Cloud Frameworks for Real-Time Healthcare Monitoring," *International Journal of Health Informatics*, vol. 22, no. 1, pp. 10-22, 2022.: https://www.journals.elsevier.com/international-journal-of-medical-informatics

[59] D. Patel et al., "Edge and Cloud Integration in Healthcare IoT Systems," *Future Generation Computer Systems*, vol. 115, pp. 457-472, 2021. https://www.journals.elsevier.com/future-generation-computer-systems

[60] Y. Wang et al., "Advances in IoT for Chronic Disease Management," *Sensors*, vol. 20, no. 23, pp. 6789-6805, 2020: https://www.mdpi.com/1424-8220/20/23/6789

[61] E. Roberts et al., "IoT-Based Healthcare Solutions for Remote Areas," *Journal of Telemedicine and Telecare*, vol. 29, no. 2, pp. 135-145, 2023. https://journals.sagepub.com/home/jtt

[62] T. Zhang et al., "Securing Data Transmission in Hybrid RPM Systems," *IEEE Transactions on Information Forensics and Security*, vol. 16, no. 8, pp. 3952-3961, 2021. https://ieeexplore.ieee.org/document/9402306

[63] R. L. Cooper et al., "Interoperability Challenges in IoT-Based Healthcare," *Journal of Biomedical Informatics*, vol. 107, pp. 103456-103472, 2020. https://www.journals.elsevier.com/journal-of-biomedical-informatics

[64] J. Singh and M. Kaur, "Efficient Data Management in IoT for Healthcare," *Computer Networks*, vol. 190, pp. 107-119, 2021. https://www.journals.elsevier.com/computer-networks

[65] Y. Zhou and L. Wang, “Analyzing the impact of COVID-19 on seasonal infectious disease transmission using SARIMAX models,” *Journal of Infection and Public Health*, vol. 16, no. 4, pp. 567–574, 2023. https://www.sciencedirect.com/science/article/pii/S1876034123000089

[66] R. Esmaeilyfard, A. K. Akbarpour, M. Shabani, and F. Keshavarz, “Enhancing AI-driven forecasting of diabetes burden,” *Scientific Reports*, vol. 15, no. 1, pp. 1–13, Jan. 2025, doi: 10.1038/s41598-024-80544-8

[67] Z. Wang, T. D. Ngo, H. Zoraghein, B. Lucas, and M. Karimzadeh, “Integrating spatiotemporal features in LSTM for spatially informed COVID-19 hospitalization forecasting,” *medRxiv*, preprint, Jan. 2025, doi: 10.1101/2025.01.13.25319973

[68] K. Kandadi and G. Shankarlingam, “Drawbacks of LSTM Algorithm: A Case Study,” SSRN, Jan. 2025. DOI: 10.2139/ssrn.5080605.

[69] P. S. Raju, “LSTM — Implementation, Advantages and Disadvantages,” *Medium*, 2023. [Online]. Available: https://medium.com/@prudhviraju.srivatsavaya/lstm-implementation-advantages-and-diadvantages-914a96fa0acb

[70] R. Kumar et al., “Serverless computing in IoT-based healthcare using AWS Lambda,” *IEEE Access*, vol. 10, pp. 102345–102357, Sep. 2022 https://ieeexplore.ieee.org/document/9900823

[71] Niu, H., Zhang, J., & Zhang, S. (2020). "Heart rate forecasting using LSTM neural networks in real-time early warning systems for intensive care units." *Journal of Healthcare Engineering*. https://www.hindawi.com/journals/jhe/2020/7243685/

[72] Zhang, X., Li, Z., & Liu, X. (2022). "Multivariate time series forecasting of vital signs in ICU patients using LSTM networks." *IEEE Transactions on Biomedical Engineering*. https://ieeexplore.ieee.org/document/9759034

[73] Cho, H., & Lee, J. (2020). "Early detection of sepsis using deep learning techniques based on patient vital signs." *Journal of Clinical Monitoring and Computing*. https://link.springer.com/article/10.1007/s10877-020-00496-1

[74] Khan, M., & Khalil, M. (2019). "Diabetes prediction and forecasting using LSTM networks." *Journal of Medical Systems*. https://link.springer.com/article/10.1007/s10916-019-1430-3

[75] M. S. Alfaris and M. O. Alfaris, "Univariate and Multivariate Time Series Blood Glucose Prediction with LSTM Deep Learning Model," Evolution in Electrical and Electronic Engineering, vol. 5, no. 1, pp. 276–285, 2024. [Online]. Available: http://eprints.uthm.edu.my/12087/1/P16855_de5069adb19c0e9bbe34a3a653fbd10c.pdf

[76] S. R. Mohanty and M. K. Baig, "Platform for precise, personalised glucose forecasting through continuous glucose and physical activity monitoring and deep learning," Medical Engineering & Physics, vol. 132, p. 104241, Oct. 2024. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/39428134/

[77] Z. Wang, X. Li, Y. Chen, and H. Zhao, "A Stacked Long Short-Term Memory Approach for Predictive Blood Glucose Monitoring in Women with Gestational Diabetes Mellitus," *Sensors*, vol. 23, no. 18, p. 7990, Sep. 2023. [Online]. Available: https://www.mdpi.com/1424-8220. Accessed: November 8, 2026.

APPENDIX “A”

Python Scripts

A1 Simulated Blood Glucose Data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter, DayLocator
from matplotlib.patches import Rectangle
from datetime import timedelta

# --- CONFIGURATION ---
start_date = '2024-01-01'
end_date = '2024-12-31 23:00'
save_path = r'C:\Python\BG_Data_JAN_DEC_2024.csv'

# Define all four plot paths to cover all 12 months
plot_path_1 = r'C:\Python\glucose_plots_Jan_Mar_2024.png'
plot_path_2 = r'C:\Python\glucose_plots_Apr_Jun_2024.png'
plot_path_3 = r'C:\Python\glucose_plots_Jul_Sep_2024.png'
plot_path_4 = r'C:\Python\glucose_plots_Oct_Dec_2024.png'

# --- Generate hourly timestamps ---
date_range = pd.date_range(start=start_date, end=end_date, freq='h')

# --- Base glucose pattern with further increased volatility ---
np.random.seed(42)
daily_rhythm = 10.0 * np.sin(2 * np.pi * date_range.hour / 24)
weekly_rhythm = 4.0 * np.sin(2 * np.pi * date_range.dayofyear / 7)
trend_noise = np.random.normal(0, 4.0, size=len(date_range))
baseline = 110 + daily_rhythm + weekly_rhythm + trend_noise
glucose_values = np.array(baseline)

# --- Meal time indicator ---
meal_time = [1 if ts.hour in [8, 13, 19] else 0 for ts in date_range]

# --- Post-meal response with increased amplitude ---
meal_kernel = np.array([1.0, 0.6, 0.4, 0.25, 0.1])
meal_effect = np.zeros_like(glucose_values)

for i, ts in enumerate(date_range):
if ts.hour in [8, 13, 19]:
amp = np.random.uniform(20.0, 25.0)
for k, w in enumerate(meal_kernel):
idx = i + k
if idx < len(meal_effect):
meal_effect[idx] += amp * w
glucose_values += meal_effect

# --- Add more frequent and wider hypo/hyper events ---
```

```
num_events_per_month = 6.5
total_months = (pd.to_datetime(end_date) - pd.to_datetime(start_date)).days / 30
num_events = int(total_months * num_events_per_month)
safe_indices = np.where(np.arange(len(date_range)) < len(date_range) - 3)[0]
random_indices = np.random.choice(safe_indices, num_events, replace=False)

for idx in random_indices:
if np.random.rand() < 0.5:
glucose_values[idx:idx+3] = np.random.uniform(40, 70, size=3)
else:
glucose_values[idx:idx+3] = np.random.uniform(180, 270, size=3)

# --- Add small surges in the NORMAL range only ---
num_normal_surges = int(len(date_range) / 80)
normal_indices = np.random.choice(len(date_range), num_normal_surges, replace=False)

for idx in normal_indices:
if glucose_values[idx] > 80 and glucose_values[idx] < 140:
glucose_values[idx:idx+2] += np.random.normal(8, 3, size=2)

# --- Add natural jitter ---
jitter = np.random.normal(0, 2.5, size=len(glucose_values))
glucose_values += jitter

# --- Clip to a more realistic and controlled range ---
glucose_values = np.clip(glucose_values, 40, 270)

# --- Light smoothing ---
glucose_series = pd.Series(glucose_values)
glucose_values = glucose_series.ewm(span=2.0, adjust=False).mean().values

# --- Build DataFrame ---
df = pd.DataFrame({
'timestamp': date_range,
'glucose': glucose_values,
'meal_time': meal_time
})

# --- Save CSV ---
df.to_csv(save_path, index=False)
print(f"✅ CSV file saved: {save_path}")

# --- Plotting function for reusability ---
def create_plot(df_to_plot, months, titles, file_path, show_plot=False):

Creates and saves a 3-month glucose plot.
If show_plot is True, the plot will remain open for display using plt.show().
if len(months) != 3:
print(f"Error: create_plot function requires exactly 3 months of data for the 3x1
layout.")
return

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(18, 12), sharey=True)
meal_colors = {'breakfast': '#FFFF00', 'lunch': '#00FFFF', 'dinner': '#00FF00'}
```

```
# Add a main title for the entire figure to indicate the month range
fig.suptitle(f'Blood Glucose Monitoring - {titles[0].split(" ")[0]} to {titles[-
1].split(" ")[0]} {titles[0].split(" ")[1]}',
fontsize=16, y=0.99)

for i, month in enumerate(months):
ax = axes[i]
month_df = df_to_plot[df_to_plot['timestamp'].dt.strftime('%Y-%m') == month]

# Plot glucose data
ax.plot(month_df['timestamp'], month_df['glucose'], color="black", linewidth=1.3)
ax.scatter(month_df.iloc[::4]['timestamp'], month_df.iloc[::4]['glucose'], color='gray',
s=10)

# Shaded background zones (Hypo/Hyper)
ax.axhspan(70, 180, color='white', alpha=0.9, zorder=0)
ax.axhspan(40, 70, color='yellow', alpha=0.25)
ax.axhspan(180, 270, color='red', alpha=0.25)

# Highlight points outside the normal range
ax.scatter(month_df[month_df['glucose'] > 180]['timestamp'],
month_df[month_df['glucose'] > 180]['glucose'], color='red', s=12)
ax.scatter(month_df[month_df['glucose'] < 70]['timestamp'], month_df[month_df['glucose']
< 70]['glucose'], color='yellow', s=12)

# Highlight meal times
for t in month_df['timestamp']:
if t.hour == 8:
ax.axvspan(t, t + timedelta(hours=1), color=meal_colors['breakfast'], alpha=0.25)
elif t.hour == 13:
ax.axvspan(t, t + timedelta(hours=1), color=meal_colors['lunch'], alpha=0.25)
elif t.hour == 19:
ax.axvspan(t, t + timedelta(hours=1), color=meal_colors['dinner'], alpha=0.25)

# Set axes limits and format
ax.set_xlim(month_df['timestamp'].min(), month_df['timestamp'].max())
ax.set_ylim(40, 270)
ax.set_yticks(np.arange(40, 271, 20))
ax.xaxis.set_major_locator(DayLocator(interval=3))
ax.xaxis.set_major_formatter(DateFormatter('%b %d'))
ax.tick_params(axis='both', labelsize=9)
ax.grid(alpha=0.3)
ax.set_ylabel('Glucose (mg/dL)', fontsize=11)

axes[-1].set_xlabel('Date', fontsize=12)

# Create legend patches
zone_patches = [Rectangle((0,0),1,1,color='yellow',alpha=0.2,label='Hypoglycemia
(<70)'),
Rectangle((0,0),1,1,color='red',alpha=0.2,label='Hyperglycemia (>180)')]
meal_patches =
[Rectangle((0,0),1,1,color=meal_colors['breakfast'],alpha=0.25,label='Breakfast'),
Rectangle((0,0),1,1,color=meal_colors['lunch'],alpha=0.25,label='Lunch'),
Rectangle((0,0),1,1,color=meal_colors['dinner'],alpha=0.25,label='Dinner')]
```

```
fig.legend(handles=zone_patches+meal_patches, loc='upper center',
bbox_to_anchor=(0.5,0.955), ncol=5, fontsize=10, frameon=True)

# Add border and save the figure
border = Rectangle((0,0),1,1, transform=fig.transFigure, fill=False, color='black',
linewidth=2)
fig.patches.append(border)

fig.tight_layout(rect=[0.04, 0, 1, 0.94])
plt.subplots_adjust(left=0.06)

# Save the file
fig.savefig(file_path, dpi=1200, bbox_inches='tight')

# Only close the figure if we don't need to show it
if not show_plot:
plt.close(fig)

# --- Create and save the four plots (12 months total) ---

# Plot 1: January to March 2024
months_1 = ['2024-01', '2024-02', '2024-03']
titles_1 = ['January 2024', 'February 2024', 'March 2024']
create_plot(df, months_1, titles_1, plot_path_1)

# Plot 2: April to June 2024
months_2 = ['2024-04', '2024-05', '2024-06']
titles_2 = ['April 2024', 'May 2024', 'June 2024']
create_plot(df, months_2, titles_2, plot_path_2)

# Plot 3: July to September 2024
months_3 = ['2024-07', '2024-08', '2024-09']
titles_3 = ['July 2024', 'August 2024', 'September 2024']
create_plot(df, months_3, titles_3, plot_path_3)

# Plot 4: October to December 2024
months_4 = ['2024-10', '2024-11', '2024-12']
titles_4 = ['October 2024', 'November 2024', 'December 2024']
create_plot(df, months_4, titles_4, plot_path_4, show_plot=False)

# The original script includes plt.show() for demonstration, but this is removed
# or commented out in the final thesis appendix version for clean presentation.
# plt.show()
```

A2 SARIMAX Forecasting Blood Glucose Data

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from matplotlib.dates import DateFormatter, DayLocator
import numpy as np
from matplotlib.patches import Rectangle
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
# --- File paths ---
data_path = r'C:\Python\BG_Data_JAN_DEC_2024.csv'
csv_save_path = r'C:\Python\glucose_forecast_Dec22_31_SARIMAX.csv'
plot_save_path = r'C:\Python\glucose_forecast_Dec22_31_SARIMAX_plot.png'

# --- Load dataset ---
data = pd.read_csv(data_path, parse_dates=['timestamp'])
data.set_index('timestamp', inplace=True)
data = data.asfreq('h')
data['glucose'] = data['glucose'].interpolate()

# --- Exogenous variables: meal_time + multiple lags for post-meal effect ---
exog = data[['meal_time']].copy()
exog['meal_lag1'] = exog['meal_time'].shift(1).fillna(0)
exog['meal_lag2'] = exog['meal_time'].shift(2).fillna(0)
exog['meal_lag3'] = exog['meal_time'].shift(3).fillna(0) # additional lag

# --- Forecast configuration ---
# Using the whole historical dataset for training as requested
train_end_date = '2024-12-21 23:00'
forecast_start = '2024-12-22'
forecast_end = '2024-12-31 23:00'
forecast_index = pd.date_range(start=forecast_start, end=forecast_end, freq='H')

train_data = data.loc[:train_end_date]
train_exog = exog.loc[:train_end_date]
exog_forecast = exog.loc[forecast_index]

# --- Fit SARIMAX with tuned parameters ---
model = SARIMAX(
    train_data['glucose'],
    exog=train_exog[['meal_time','meal_lag1','meal_lag2','meal_lag3']],
    order=(2,1,2),
    seasonal_order=(1,1,1,24),
    trend='c',
    enforce_stationarity=False,
    enforce_invertibility=False
)
results = model.fit(disp=False)
print("✅ Tuned SARIMAX model fitted")
print(results.summary())

# --- Forecast ---
forecast_steps = len(forecast_index)
forecast = results.get_forecast(steps=forecast_steps,

exog=exog_forecast[['meal_time','meal_lag1','meal_lag2','meal_lag3']])
forecast_mean = forecast.predicted_mean.clip(65, 250)
forecast_mean = forecast_mean.rolling(2, min_periods=1).mean()

# --- Confidence intervals ---
conf_int = forecast.conf_int(alpha=0.2)
conf_int['lower'] = conf_int.iloc[:,0].clip(50,250)
conf_int['upper'] = conf_int.iloc[:,1].clip(50,250)
```

```
# --- Save forecast CSV ---
forecast_df = pd.DataFrame({
    'timestamp': forecast_index,
    'forecast_glucose': forecast_mean.values.round(2)
})
forecast_df.to_csv(csv_save_path, index=False)
print(f"✅ Tuned SARIMAX forecast CSV saved: {csv_save_path}")

# --- Calculate MAE and RMSE ---
# Retrieve actual glucose values for the forecast period to compare with predictions
actual_glucose = data.loc[forecast_index, 'glucose']
actual_glucose.dropna(inplace=True)
forecast_mean_aligned = forecast_mean.reindex(actual_glucose.index)

# Calculate metrics in mg/dL
mae = mean_absolute_error(actual_glucose, forecast_mean_aligned)
rmse = np.sqrt(mean_squared_error(actual_glucose, forecast_mean_aligned))

# Calculate metrics in percentage
mean_actual_glucose = actual_glucose.mean()
mae_percent = (mae / mean_actual_glucose) * 100
rmse_percent = (rmse / mean_actual_glucose) * 100

print(f"\n✅ Forecast Metrics for Dec 22-31, 2024:")
print(f"   Mean Absolute Error (MAE): {mae:.2f} mg/dL  | {mae_percent:.2f}%")
print(f"   Root Mean Squared Error (RMSE): {rmse:.2f} mg/dL | {rmse_percent:.2f}%")

# --- Plot Historical + Forecast ---
plt.figure(figsize=(22,7))

# Glucose ranges
plt.axhspan(50, 70, color='yellow', alpha=0.15)
plt.axhspan(70, 180, color='lightgreen', alpha=0.08)
plt.axhspan(180, 300, color='red', alpha=0.12)

# Historical line (Dec 10–21)
hist_data = data.loc['2024-12-10':'2024-12-21']
plt.plot(hist_data.index, hist_data['glucose'], color='green', linewidth=1.8,
label='Historical (Dec 10–21)')

# Forecast line (Dec 22–31)
plt.plot(forecast_index, forecast_mean, color='black', linewidth=2.2, label='SARIMAX
Forecast (Dec 22–31)')

# Confidence interval
plt.fill_between(forecast_index, conf_int['lower'], conf_int['upper'], color='gray',
alpha=0.3, label='80% Confidence Interval')

# Formatting
plt.axvline(forecast_index[0], color='gray', linestyle='--', alpha=0.6)
plt.xlim(pd.Timestamp('2024-12-10'), pd.Timestamp('2024-12-31 23:00'))
plt.ylim(50,250)
plt.grid(alpha=0.2)
```

```
plt.title('December 10-31, 2024: Glucose Forecast using SARIMAX Model', fontsize=15,
pad=15)
plt.ylabel('Glucose (mg/dL)', fontsize=12)

# X-axis formatting
ax = plt.gca()
ax.xaxis.set_major_locator(DayLocator(interval=2))
ax.xaxis.set_major_formatter(DateFormatter('%b %d\n%a'))
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.legend(loc='upper center', ncol=3, frameon=True, fontsize=10)

# Add border around the plot
border = Rectangle((0, 0), 1, 1, transform=plt.gcf().transFigure, fill=False,
edgecolor='black', linewidth=2)
plt.gcf().patches.append(border)

# Save plot
plt.savefig(plot_save_path, dpi=1200, bbox_inches='tight')
# The original script includes plt.show() for demonstration, but this is removed
# or commented out in the final thesis appendix version for clean presentation.
# plt.show()
print(f"✅ Tuned SARIMAX forecast plot saved: {plot_save_path}")
```

A3 Historical & SARIMAX Forecasted Data Comparison

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter, DayLocator
import numpy as np
from matplotlib.patches import Rectangle
from sklearn.metrics import mean_absolute_error, mean_squared_error

# --- File paths ---
ACTUAL_PATH = r'C:\Python\BG_Data_JAN_DEC_2024.csv'
FORECAST_PATH = r'C:\Python\glucose_forecast_Dec22_31_SARIMAX.csv'
OUTPUT_CSV = r'C:\Python\glucose_comparison_SARIMAX_Dec22_31.csv'
OUTPUT_PLOT = r'C:\Python\Comparison_SARIMAX_vs_Actual_Dec_Styled.png'

# --- Load actual glucose data (Dec 22–31) ---
actual = pd.read_csv(ACTUAL_PATH, parse_dates=['timestamp'])
actual.set_index('timestamp', inplace=True)
actual['actual_glucose'] = actual['glucose'].interpolate()
actual = actual.loc['2024-12-22':'2024-12-31', ['actual_glucose']]

# --- Load forecasted SARIMAX glucose data (Dec 22–31) ---
forecast = pd.read_csv(FORECAST_PATH, parse_dates=['timestamp'])
forecast.set_index('timestamp', inplace=True)
forecast.rename(columns={'forecast_glucose': 'forecasted_glucose'}, inplace=True)
forecast = forecast.loc['2024-12-22':'2024-12-31', ['forecasted_glucose']]

# --- Combine actual and forecast, compute errors ---
df = actual.join(forecast, how='inner')
```

```
df['difference'] = df['forecasted_glucose'] - df['actual_glucose']
df['abs_diff'] = df['difference'].abs()
df['error_percent'] = (df['abs_diff'] / df['actual_glucose']) * 100

# --- Calculate MAE and RMSE ---
mae = df['abs_diff'].mean()
rmse = np.sqrt((df['difference']**2).mean())
mae_percent = df['error_percent'].mean()
rmse_percent = np.sqrt((df['error_percent']**2).mean())

# --- Save comparison CSV ---
df.reset_index(inplace=True)
df.to_csv(OUTPUT_CSV, index=False)
print(f"✅ Comparison CSV saved: {OUTPUT_CSV}")

# --- Plot Actual vs SARIMAX Forecast with Error % ---
fig, ax1 = plt.subplots(figsize=(16, 7))

# Bright colors for visibility
line_actual = ax1.plot(df['timestamp'], df['actual_glucose'], color='dodgerblue',
linewidth=2.5, label='Actual Glucose')
line_forecast = ax1.plot(df['timestamp'], df['forecasted_glucose'], color='orange',
linewidth=2.5, linestyle='--', label='SARIMAX Forecast')

ax1.set_ylabel('Glucose (mg/dL)', fontsize=12)
ax1.grid(alpha=0.3)

# Secondary y-axis for percent error (bright red)
ax2 = ax1.twinx()
line_error = ax2.plot(df['timestamp'], df['error_percent'], color='red', linestyle=':',
linewidth=2, label='Error (%)')
ax2.set_ylabel('Error (%)', color='red', fontsize=12)
ax2.tick_params(axis='y', labelcolor='red')
ax2.set_ylim(0, df['error_percent'].max() * 1.2)

# Title & Axis Formatting
plt.title('December 22–31, 2024: Actual vs SARIMAX Forecast & Error %', fontsize=16,
pad=15)
ax1.xaxis.set_major_locator(DayLocator())
ax1.xaxis.set_major_formatter(DateFormatter('%b %d\n%a'))
plt.xticks(rotation=45, fontsize=11)
plt.yticks(fontsize=11)

# Legend
handles = line_actual + line_forecast + line_error
labels = [h.get_label() for h in handles]
ax1.legend(handles, labels, loc='upper center', bbox_to_anchor=(0.5, 0.95), ncol=3,
fontsize=11)

# Display MAE and RMSE inside plot (top-left)
metrics_text = (
    f'MAE:  {mae:.2f} mg/dL ({mae_percent:.2f}%)\n'
    f'RMSE: {rmse:.2f} mg/dL ({rmse_percent:.2f}%)'
)
ax1.text(0.02, 0.90, metrics_text,
```

```
        transform=ax1.transAxes,
        fontsize=12, color='black',
        bbox=dict(facecolor='white', alpha=0.7, edgecolor='black'))

# --- Add border around the plot ---
fig.patches.append(Rectangle((0, 0), 1, 1, transform=fig.transFigure, fill=False,
edgecolor='black', linewidth=2))

# Save and show plot
plt.tight_layout()
plt.savefig(OUTPUT_PLOT, dpi=1200, bbox_inches='tight')
# The original script includes plt.show() for demonstration, but this is removed
# or commented out in the final thesis appendix version for clean presentation.
# plt.show()
print(f"✅ Comparison plot saved: {
```

A4 LSTM Forecasting Blood Glucose Data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from matplotlib.dates import DateFormatter, DayLocator
from matplotlib.patches import Rectangle
from sklearn.metrics import mean_absolute_error

# --- File paths ---
data_path = r'C:\Python\BG_Data_JAN_DEC_2024.csv'
csv_save_path = r'C:\Python\glucose_forecast_LSTM_Dec22_31_2024_BIDIRECTIONAL.csv'
plot_save_path = r'C:\Python\glucose_forecast_LSTM_Dec22_31_2024_BIDIRECTIONAL_plot.png'

# --- Define Time Periods ---
train_end_date = '2024-12-21 23:00'
forecast_start_date = '2024-12-22 00:00'
forecast_end_date = '2024-12-31 23:00'

# --- Load and preprocess data ---
data = pd.read_csv(data_path, parse_dates=['timestamp'])
data.set_index('timestamp', inplace=True)
data = data.asfreq('h')
data['glucose'] = data['glucose'].interpolate()
data['hour'] = data.index.hour
data['meal_time'] = data['hour'].isin([8, 13, 19]).astype(int)

# --- Feature Selection for Model ---
df = data[['glucose', 'meal_time']].copy()

# Scale the data before creating sequences
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df.loc[:train_end_date])
```

```
df_scaled = pd.DataFrame(scaled_data, columns=df.columns,
index=df.loc[:train_end_date].index)

# --- Prepare data for LSTM ---
def create_sequences(data_scaled_df, window_size):
    X, y = [], []
    train_data_len = len(data_scaled_df)
    for i in range(train_data_len - window_size):
        X.append(data_scaled_df.iloc[i: i + window_size].values)
        y.append(data_scaled_df.iloc[i + window_size]['glucose'])
    return np.array(X), np.array(y)

WINDOW_SIZE = 24
X, y = create_sequences(df_scaled, WINDOW_SIZE)
X_train, y_train = X, y

# --- Build and Train a BIDIRECTIONAL LSTM Model ---
model = Sequential()
model.add(Bidirectional(LSTM(512, activation='relu', return_sequences=True),
input_shape=(WINDOW_SIZE, X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(256, activation='relu', return_sequences=False)))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

print("Starting LSTM model training (with a Bidirectional architecture)...")
model.fit(X_train, y_train, epochs=40, batch_size=32, verbose=1)
print("✅ LSTM model trained.")

# --- Rolling Forecast Implementation ---
forecast_scaled_results = []
forecast_index_range = pd.date_range(start=forecast_start_date, end=forecast_end_date,
freq='H')

last_known_historical_data = df.loc[:train_end_date].iloc[-WINDOW_SIZE:]
current_input_seq = scaler.transform(last_known_historical_data).reshape(1, WINDOW_SIZE,
X_train.shape[2])

future_meal_time = data.loc[forecast_start_date:forecast_end_date]['meal_time'].values

for i in range(len(forecast_index_range)):
    predicted_glucose_scaled = model.predict(current_input_seq, verbose=0)[0][0]

    forecast_scaled_results.append(predicted_glucose_scaled)

    next_meal_time_scaled = scaler.transform([[0, future_meal_time[i]]])[0, 1]

    new_input_row = np.array([[predicted_glucose_scaled,
next_meal_time_scaled]]).reshape(1, 1, X_train.shape[2])

    current_input_seq = np.append(current_input_seq[:, 1:, :], new_input_row, axis=1)

forecast_scaled_results = np.array(forecast_scaled_results).squeeze()
```

```
predicted_glucose_unscaled =
scaler.inverse_transform(np.hstack((forecast_scaled_results.reshape(-1, 1),
np.zeros((forecast_scaled_results.shape[0], df.shape[1]-1)))))[:, 0]

# --- Prepare forecast DataFrame ---
forecast_df = pd.DataFrame({
    'timestamp': forecast_index_range,
    'forecast_glucose': predicted_glucose_unscaled.round(2)
})
forecast_df.to_csv(csv_save_path, index=False)
print(f"✅ Forecast CSV saved: {csv_save_path}")

# --- Calculate MAE for the forecast period ---
actual_data_forecast_period = data.loc[forecast_start_date:forecast_end_date, 'glucose']
forecast_mae = mean_absolute_error(actual_data_forecast_period,
predicted_glucose_unscaled)
print(f"✅ Calculated MAE for forecast period: {forecast_mae:.2f} mg/dL")
print(f"✅ Calculated MAE as a percentage: {(forecast_mae /
actual_data_forecast_period.mean()) * 100:.2f}%")

# --- Plotting ---
plt.figure(figsize=(22, 7))
plt.axhspan(50, 70, color='yellow', alpha=0.15, label='Hypoglycemic (50-70 mg/dL)')
plt.axhspan(70, 180, color='lightgreen', alpha=0.08, label='Normal (70-180 mg/dL)')
plt.axhspan(180, 300, color='red', alpha=0.12, label='Hyperglycemic (>180 mg/dL)')

historical_plot_start = '2024-12-10 00:00'
historical_plot_end = '2024-12-21 23:00'
historical_to_plot = df.loc[historical_plot_start:historical_plot_end]
plt.plot(historical_to_plot.index, historical_to_plot['glucose'], color='green',
linewidth=1.8, label=f'Historical ({pd.to_datetime(historical_plot_start).strftime("%b
%d")}–{pd.to_datetime(historical_plot_end).strftime("%b %d")})')

plt.plot(forecast_index_range, predicted_glucose_unscaled, color='black', linewidth=2.2,
label=f'LSTM Forecast ({pd.to_datetime(forecast_start_date).strftime("%b %d")}–
{pd.to_datetime(forecast_end_date).strftime("%b %d")})')

plt.axvline(pd.Timestamp(forecast_start_date), color='gray', linestyle='--', alpha=0.6)
plt.xlim(pd.Timestamp(historical_plot_start), pd.Timestamp(forecast_end_date))
plt.ylim(40, 260)
plt.grid(alpha=0.2)
plt.title(f'December 10–31, 2024: Rolling Glucose Forecast Using Bidirectional LSTM
Model', fontsize=15, pad=15)
plt.ylabel('Glucose (mg/dL)', fontsize=12)

ax = plt.gca()
ax.xaxis.set_major_locator(DayLocator(interval=2))
ax.xaxis.set_major_formatter(DateFormatter('%b %d\n%a'))
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.legend(loc='upper center', ncol=4, frameon=True, fontsize=10)

fig = plt.gcf()
```

```
border = Rectangle((0, 0), 1, 1, transform=fig.transFigure, fill=False,
edgecolor='black', linewidth=2)
fig.patches.append(border)

plt.tight_layout()
plt.savefig(plot_save_path, dpi=1200, bbox_inches='tight')
# The original script includes plt.show() for demonstration, but this is removed
# or commented out in the final thesis appendix version for clean presentation.
# plt.show()
print(f"✅ Forecast plot saved: {plot_save_path}")
```

A5 Historical & LSTM Forecasted Data Comparison

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter, DayLocator
import numpy as np
from matplotlib.patches import Rectangle
from sklearn.metrics import mean_absolute_error, mean_squared_error

# --- File paths ---
ACTUAL_PATH = r'C:\Python\BG_Data_JAN_DEC_2024.csv'
FORECAST_PATH = r'C:\Python\glucose_forecast_LSTM_Dec22_31_2024_BIDIRECTIONAL.csv'
OUTPUT_CSV = r'C:\Python\glucose_comparison_LSTM_Dec22_31.csv'
OUTPUT_PLOT = r'C:\Python\Comparison_LSTM_vs_Actual_Dec_Styled.png'

# --- Load actual glucose data (Dec 10–31) ---
actual = pd.read_csv(ACTUAL_PATH, parse_dates=['timestamp'])
actual.set_index('timestamp', inplace=True)
actual['actual_glucose'] = actual['glucose'].interpolate()
actual = actual.loc['2024-12-10':'2024-12-31', ['actual_glucose']] # Adjusted date range

# --- Load forecasted LSTM glucose data (Dec 22–31) ---
forecast = pd.read_csv(FORECAST_PATH, parse_dates=['timestamp'])
forecast.set_index('timestamp', inplace=True)
forecast.rename(columns={'forecast_glucose': 'forecasted_glucose'}, inplace=True)
forecast = forecast.loc['2024-12-22':'2024-12-31', ['forecasted_glucose']] # Adjusted
date range

# --- Combine actual and forecast, compute errors ---
df = actual.join(forecast, how='inner')
df['difference'] = df['forecasted_glucose'] - df['actual_glucose']
df['abs_diff'] = df['difference'].abs()
df['error_percent'] = (df['abs_diff'] / df['actual_glucose']) * 100


# --- Calculate MAE and RMSE ---
mae = df['abs_diff'].mean()
rmse = np.sqrt((df['difference']**2).mean())
mae_percent = df['error_percent'].mean()
rmse_percent = np.sqrt((df['error_percent']**2).mean())

# --- Save comparison CSV ---
```

```
df.reset_index(inplace=True)
df.to_csv(OUTPUT_CSV, index=False)
print(f"✅ Comparison CSV saved: {OUTPUT_CSV}")

# --- Plot Actual vs LSTM Forecast with Error % ---
fig, ax1 = plt.subplots(figsize=(16, 7))

# Bright colors for visibility
line_actual = ax1.plot(df['timestamp'], df['actual_glucose'], color='dodgerblue',
linewidth=2.5, label='Actual Glucose')
line_forecast = ax1.plot(df['timestamp'], df['forecasted_glucose'], color='orange',
linewidth=2.5, linestyle='--', label='LSTM Forecast')

ax1.set_ylabel('Glucose (mg/dL)', fontsize=12)
ax1.grid(alpha=0.3)

# Secondary y-axis for percent error (bright red)
ax2 = ax1.twinx()
line_error = ax2.plot(df['timestamp'], df['error_percent'], color='red', linestyle=':',
linewidth=2, label='Error (%)')
ax2.set_ylabel('Error (%)', color='red', fontsize=12)
ax2.tick_params(axis='y', labelcolor='red')
ax2.set_ylim(0, df['error_percent'].max() * 1.2)

# Title & Axis Formatting
plt.title('December 10–31, 2024: Actual vs Bidirectional LSTM Forecast & Error %',
fontsize=16, pad=15)
ax1.xaxis.set_major_locator(DayLocator())
ax1.xaxis.set_major_formatter(DateFormatter('%b %d\n%a'))
plt.xticks(rotation=45, fontsize=11)
plt.yticks(fontsize=11)

# Legend
handles = line_actual + line_forecast + line_error
labels = [h.get_label() for h in handles]
ax1.legend(handles, labels, loc='upper center', bbox_to_anchor=(0.5, 0.95), ncol=3,
fontsize=11)

# Display MAE and RMSE inside plot (top-left)
metrics_text = (
    f'MAE:  {mae:.2f} mg/dL ({mae_percent:.2f}%)\n'
    f'RMSE: {rmse:.2f} mg/dL ({rmse_percent:.2f}%)'
)
ax1.text(0.02, 0.90, metrics_text,
          transform=ax1.transAxes,
          fontsize=12, color='black',
          bbox=dict(facecolor='white', alpha=0.7, edgecolor='black'))

# --- Add border around the plot ---
fig = plt.gcf()
border = Rectangle((0, 0), 1, 1, transform=fig.transFigure, fill=False,
edgecolor='black', linewidth=2)
fig.patches.append(border)

# Save and show plot
```

```
plt.tight_layout()
plt.savefig(OUTPUT_PLOT, dpi=1200, bbox_inches='tight')
# The original script includes plt.show() for demonstration, but this is removed
# or commented out in the final thesis appendix version for clean presentation.
# plt.show()
print(f"✅ Comparison plot saved: {OUTPUT_PLOT}")
```

A6 Performance & Accuracy Comparison SARIMAX & LSTM

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter, DayLocator
import numpy as np
from matplotlib.patches import Rectangle
from sklearn.metrics import mean_absolute_error, mean_squared_error

# --- File paths ---
ACTUAL_PATH = r'C:\Python\BG_Data_JAN_DEC_2024.csv'
SARIMAX_FORECAST_PATH = r'C:\Python\glucose_forecast_Dec22_31_SARIMAX.csv'
LSTM_FORECAST_PATH = r'C:\Python\glucose_forecast_LSTM_Dec22_31_2024_BIDIRECTIONAL.csv'
OUTPUT_CSV = r'C:\Python\glucose_comparison_SARIMAX_vs_LSTM_Dec22_31.csv'
OUTPUT_PLOT = r'C:\Python\Comparison_SARIMAX_vs_LSTM_Dec_Styled.png'

# --- Define Date Range ---
START_DATE = '2024-12-22'
END_DATE = '2024-12-31'

# --- Load actual glucose data ---
actual = pd.read_csv(ACTUAL_PATH, parse_dates=['timestamp'])
actual.set_index('timestamp', inplace=True)
actual['actual_glucose'] = actual['glucose'].interpolate()
actual = actual.loc[START_DATE:END_DATE, ['actual_glucose']]

# --- Load LSTM forecast data and compute errors ---
lstm_forecast = pd.read_csv(LSTM_FORECAST_PATH, parse_dates=['timestamp'])
lstm_forecast.set_index('timestamp', inplace=True)
lstm_forecast.rename(columns={'forecast_glucose': 'forecasted_lstm'}, inplace=True)
lstm_forecast = lstm_forecast.loc[START_DATE:END_DATE, ['forecasted_lstm']]

lstm_df = actual.join(lstm_forecast, how='inner')
lstm_mae = mean_absolute_error(lstm_df['actual_glucose'], lstm_df['forecasted_lstm'])
lstm_rmse = np.sqrt(mean_squared_error(lstm_df['actual_glucose'],
lstm_df['forecasted_lstm']))
lstm_mae_percent = (lstm_mae / lstm_df['actual_glucose'].mean()) * 100
lstm_rmse_percent = (lstm_rmse / lstm_df['actual_glucose'].mean()) * 100

# --- Load SARIMAX forecast data and compute errors ---
sarimax_forecast = pd.read_csv(SARIMAX_FORECAST_PATH, parse_dates=['timestamp'])
sarimax_forecast.set_index('timestamp', inplace=True)
sarimax_forecast.rename(columns={'forecast_glucose': 'forecasted_sarimax'},
inplace=True)
sarimax_forecast = sarimax_forecast.loc[START_DATE:END_DATE, ['forecasted_sarimax']]
```

```
sarimax_df = actual.join(sarimax_forecast, how='inner')
sarimax_mae = mean_absolute_error(sarimax_df['actual_glucose'],
sarimax_df['forecasted_sarimax'])
sarimax_rmse = np.sqrt(mean_squared_error(sarimax_df['actual_glucose'],
sarimax_df['forecasted_sarimax']))
sarimax_mae_percent = (sarimax_mae / sarimax_df['actual_glucose'].mean()) * 100
sarimax_rmse_percent = (sarimax_rmse / sarimax_df['actual_glucose'].mean()) * 100

# --- Combine all data for plotting and CSV export ---
df_combined = actual.join(lstm_forecast, how='inner').join(sarimax_forecast,
how='inner')
df_combined.reset_index(inplace=True)
df_combined.to_csv(OUTPUT_CSV, index=False)
print(f"✅ Comparison CSV saved: {OUTPUT_CSV}")

# --- Plot Actual vs SARIMAX vs LSTM Forecast ---
fig, ax1 = plt.subplots(figsize=(16, 7))

# Plot all three lines with attractive colors
ax1.plot(df_combined['timestamp'], df_combined['actual_glucose'], color='royalblue',
linewidth=3, label='Actual Glucose')
ax1.plot(df_combined['timestamp'], df_combined['forecasted_lstm'], color='orange',
linewidth=2.5, linestyle='--', label='LSTM Forecast')
ax1.plot(df_combined['timestamp'], df_combined['forecasted_sarimax'], color='limegreen',
linewidth=2.5, linestyle=':', label='SARIMAX Forecast')

ax1.set_ylabel('Glucose (mg/dL)', fontsize=12)
ax1.grid(alpha=0.3)
ax1.set_ylim(df_combined[['actual_glucose', 'forecasted_lstm',
'forecasted_sarimax']].min().min() - 10,
             df_combined[['actual_glucose', 'forecasted_lstm',
'forecasted_sarimax']].max().max() + 10)

# Title & Axis Formatting
plt.title(f'{START_DATE}–{END_DATE}: LSTM vs. SARIMAX Forecast Comparison', fontsize=16,
pad=15)
ax1.xaxis.set_major_locator(DayLocator())
ax1.xaxis.set_major_formatter(DateFormatter('%b %d\n%a'))
plt.xticks(rotation=45, fontsize=11)
plt.yticks(fontsize=11)

# Legend
ax1.legend(loc='upper center', bbox_to_anchor=(0.5, 0.95), ncol=3, fontsize=11)

# Display MAE and RMSE for each model
sarimax_metrics_text = (
    f'SARIMAX Metrics:\n'
    f'MAE:  {sarimax_mae:.2f} mg/dL ({sarimax_mae_percent:.2f}%)\n'
    f'RMSE: {sarimax_rmse:.2f} mg/dL ({sarimax_rmse_percent:.2f}%)'
)
ax1.text(0.98, 0.95, sarimax_metrics_text,
         transform=ax1.transAxes,
         fontsize=10, color='black', ha='right', va='top',
         bbox=dict(facecolor='white', alpha=0.7, edgecolor='black'))
```

```
lstm_metrics_text = (
    f'LSTM Metrics:\n'
    f'MAE:  {lstm_mae:.2f} mg/dL ({lstm_mae_percent:.2f}%)\n'
    f'RMSE: {lstm_rmse:.2f} mg/dL ({lstm_rmse_percent:.2f}%)'
)
ax1.text(0.02, 0.95, lstm_metrics_text,
         transform=ax1.transAxes,
         fontsize=10, color='black', ha='left', va='top',
         bbox=dict(facecolor='white', alpha=0.7, edgecolor='black'))

# Add border around the plot
fig.patches.append(Rectangle((0, 0), 1, 1, transform=fig.transFigure, fill=False,
edgecolor='black', linewidth=2))

# Save and show plot
plt.tight_layout()
plt.savefig(OUTPUT_PLOT, dpi=1200, bbox_inches='tight')
plt.show()
print(f"✅ Comparison plot saved: {OUTPUT_PLOT}")
```

A7 Data Uploader From Microcontroller To AWS S3 Bucket

```
import boto3
from botocore.exceptions import NoCredentialsError, ClientError
import os

# --- 1. CONFIGURATION: UPDATE THIS LINE ---
# CHANGE 'your-unique-bucket-name' to the exact name of your S3 bucket.
BUCKET_NAME = 'wasif-pi-project-280925'

# --- 2. FILES DEFINITION ---
# This list specifies the local path on your Raspberry Pi and the desired path/name on
S3.
FILES_TO_UPLOAD = [
{
'local_path': '/home/wasif/BG_Data_JAN_DEC_2024.csv',
's3_key': 'historical_data/BG_Data_JAN_DEC_2024.csv'  # Creates 'historical_data' folder
on S3
},
{
'local_path': '/home/wasif/glucose_forecast_Dec22_31_SARIMAX.csv',
's3_key': 'forecast_data/glucose_forecast_Dec22_31_SARIMAX.csv'  # Creates
'forecast_data' folder on S3
}
]

def upload_to_s3(local_file, bucket, s3_file):
"""Uploads a single file to an AWS S3 bucket using configured credentials."""
try:
# Initialize S3 client (uses credentials from ~/.aws/credentials)
s3 = boto3.client('s3')

# Check if the file exists locally on the Pi
```

```
if not os.path.exists(local_file):
print(f"--- FAILED: Local file NOT FOUND at {local_file}")
return False

print(f"Uploading {local_file.split('/')[-1]} to s3://{bucket}/{s3_file}...")

# Perform upload
s3.upload_file(local_file, bucket, s3_file)

print(f"--- SUCCESS: {s3_file} uploaded successfully.")
return True

except NoCredentialsError:
print("\n--- FATAL ERROR: AWS credentials not found. Please check ~/.aws/credentials.")
return False

except ClientError as e:
# Handles errors like invalid bucket, region mismatch, or permission denied
print(f"--- ERROR: S3 Client Error: {e}")
return False

except Exception as e:
print(f"--- UNEXPECTED ERROR: {e}")
return False


if __name__ == "__main__":
print(f"Starting upload process to S3 bucket: {BUCKET_NAME}\n")

all_successful = True
for file_info in FILES_TO_UPLOAD:
success = upload_to_s3(file_info['local_path'], BUCKET_NAME, file_info['s3_key'])
if not success:
all_successful = False

if all_successful:
print("\n✅ All files successfully uploaded!")
else:
print("\n⚠ One or more files failed to upload. Review the errors above.")
```

A8 Configuring Raspberry PI 5 as MODBUS Server Device Over TCP/IP Network

```
# --- Raspberry Pi as Modbus TCP Server (Slave) ---
# Make sure pymodbus is installed: pip install pymodbus

from pymodbus.server import StartTcpServer
from pymodbus.device import ModbusDeviceIdentification
from pymodbus.datastore import ModbusSequentialDataBlock, ModbusSlaveContext,
ModbusServerContext
from pymodbus.version import version
from datetime import datetime
```

```
import random
import logging

# Enable basic logging (optional)
logging.basicConfig(format='%(asctime)s %(message)s', level=logging.INFO)

# --- 1. DEFINE INITIAL DATA BLOCKS ---
# These simulate sensor values (e.g., heart rate, temperature, etc.)
store = ModbusSlaveContext(
di=ModbusSequentialDataBlock(0, [17]*100),   # Discrete Inputs
co=ModbusSequentialDataBlock(0, [0]*100),    # Coils
hr=ModbusSequentialDataBlock(0, [0]*100),    # Holding Registers
ir=ModbusSequentialDataBlock(0, [0]*100)     # Input Registers
)
context = ModbusServerContext(slaves=store, single=True)

# --- 2. DEVICE INFORMATION ---
identity = ModbusDeviceIdentification()
identity.VendorName = 'RaspberryPi'
identity.ProductCode = 'RPi5'
identity.VendorUrl = 'https://www.raspberrypi.com/'
identity.ProductName = 'Pi Modbus Server'
identity.ModelName = 'PiModbusV1'
identity.MajorMinorRevision = version.short()

# --- 3. FUNCTION TO SIMULATE DATA UPDATES ---
def update_registers(context):
"""
Simulates sensor data like heart rate, temperature, etc.
"""
hr = random.randint(60, 100)     # heart rate
temp = random.randint(36, 38)    # temperature
glucose = random.randint(90, 130)

# Write to holding registers
context[0].setValues(3, 0, [hr, temp, glucose])
logging.info(f"Updated HR={hr}, Temp={temp}, Glucose={glucose} at {datetime.now()}")

# --- 4. START MODBUS TCP SERVER ---
from threading import Thread
import time

def updater():
while True:
time.sleep(5)
update_registers(context)

Thread(target=updater, daemon=True).start()

# --- 5. RUN SERVER ---
if __name__ == "__main__":
logging.info("Starting Modbus TCP Server on 0.0.0.0:5020 ...")
StartTcpServer(context, identity=identity, address=("0.0.0.0", 5020))
```