PENTHUAURA: DESIGN AND IMPLEMENTATION OF A NON-INVASIVE IOT HEALTH MONITORING SYSTEM WITH MACHINE LEARNING FOR ADVANCED VITALS PREDICTION

By
Diaa Ahmed Hussien
Yehia Mohamed Mahmoud
Mohanad Tarek Saad
Abdulrahman Nageh Abdulrahman

Under Supervision of
Dr. Omar Mahmoud Sabry

A Thesis Submitted to
Canadian International College – Zayed Campus

in Partial Fulfillment of the
Requirements for the Degree of Bachelor in
Communications and Electronics Engineering

CANADIAN INTERNATIONAL COLLEGE
GIZA, EGYPT
2025

Declaration

We, the undersigned, declare that this thesis, titled "PenthuAura: Design and Implementation of a Non-Invasive IoT Health Monitoring System with Machine Learning for Advanced Vitals Prediction," is the result of our own original, collaborative research work. All sources of information have been specifically acknowledged by means of references. This work has not been submitted, in whole or in part, for any other degree or qualification at this or any other university.

Team Members,

Name	Signature
Diaa Ahmed Hussien	
Yehia Mohamed Mahmoud	
Mohanad Tarek Saad	
Abdulrahman Nageh Abdulrahman	

Acknowledgments

We would like to express our deepest collective gratitude to our supervisor, Dr. Omar Mahmoud Sabry, for their invaluable guidance, unwavering support, and insightful feedback throughout this project. Their expertise was instrumental in steering our team's efforts and in navigating the complexities of both the hardware design and the machine learning pipeline. Their open-door policy and willingness to engage in detailed technical discussions were crucial to our success.

Our sincere thanks also go to the faculty and staff of the Department of Communications and Electronics Engineering for providing the foundational knowledge and resources that made this ambitious project possible. The lab facilities and the academic environment were essential for the practical implementation and testing of our system.

We are also incredibly grateful for the support of our families and friends, whose encouragement and belief in our work have been a constant source of motivation for the entire team.

Finally, we wish to acknowledge the seamless collaboration and dedication of every member of this team. This project was a true synthesis of different skills and perspectives, and its success is a testament to what can be achieved through shared vision and mutual respect.

Table of Contents

List of Tables

List of Figures

Glossary of Terms / List of Abbreviations

ABP	Arterial Blood Pressure
AC	Alternating Current (in context of signals, the pulsatile component)
ANS	Autonomic Nervous System
API	Application Programming Interface
BOM	Bill of Materials
BP	Blood Pressure
DBP	Diastolic Blood Pressure
DC	Direct Current (in context of signals, the non-pulsatile baseline)
DFA	Detrended Fluctuation Analysis
ECG	Electrocardiogram
ESP32	A low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth.
Flask	A lightweight web application framework for Python
GPIO	General-Purpose Input/Output
HR	Heart Rate
HRV	Heart Rate Variability
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit (a serial communication protocol)
IoMT	Internet of Medical Things
IoT	Internet of Things
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
MAE	Mean Absolute Error

ML	Machine Learning
Ngrok	A reverse proxy service that creates a secure tunnel from a public endpoint to a locally running web service.
NIBP	Non-Invasive Blood Pressure
OTA	Over-The-Air
PCB	Printed Circuit Board
PPG	Photoplethysmography
PSD	Power Spectral Density
PTT	Pulse Transit Time
RAM	Random Access Memory
REST	Representational State Transfer
RMSE	Root Mean Squared Error
RPi	Raspberry Pi
RTC	Real-Time Clock
SBP	Systolic Blood Pressure
SMOTE	Synthetic Minority Over-sampling Technique
SpO2	Peripheral Capillary Oxygen Saturation
systemd	A system and service manager for Linux operating systems.
ThingSpeak	An IoT analytics platform service that allows for logging and retrieving data over the internet using an HTTP API.
UI	User Interface

Abstract

The rising prevalence of chronic health conditions necessitates a shift towards proactive and accessible health monitoring. This thesis details the design, implementation, and evaluation of "PenthuAura," a novel, low-cost Internet of Things (IoT) health monitoring system developed through a collaborative team effort to non-invasively predict advanced physiological metrics. The system's methodology involved a distributed architecture where an ESP32-based edge device integrated a MAX30102 sensor for photoplethysmography (PPG) and a MAX30205 for temperature sensing. This device acquired raw physiological data and transmitted it to a Raspberry Pi backend server via a robust communication pipeline established using ThingSpeak and Ngrok. The backend server hosted a sophisticated machine learning pipeline to process the raw PPG data, predicting Systolic and Diastolic Blood Pressure, user stress levels, and an overall health state. This was achieved using a Random Forest Regressor for blood pressure, Random Forest Classifiers for stress and health state, and an Isolation Forest for anomaly detection. A critical contribution of this work is the data processing strategy where both clinical training data (from VitalDB) and live sensor data were standardized through baseline correction, amplitude scaling, and bandpass filtering to ensure a consistent data format. The dataset labels for the classification models were meticulously applied based on an extensive review of medical literature. The implemented system was successfully tested end-to-end, demonstrating reliable data transmission and processing. The PenthuAura project serves as a successful proof-of-concept, showcasing the potential of integrating low-cost IoT hardware with a carefully developed machine learning pipeline to create powerful, accessible tools for personal health guardianship.

Keywords: Internet of Things, health monitoring, photoplethysmography, machine learning, cuffless blood pressure, Random Forest, ESP32, signal processing

Chapter (1)

Introduction

Chapter 1 : Introduction

1.1. Background and Motivation

In the 21st century, healthcare is undergoing a paradigm shift, moving from a reactive, episode-based model to one that is proactive, personalized, and preventative. The global rise in chronic diseases such as hypertension, cardiovascular disorders, and stress-related conditions has underscored the limitations of traditional, clinical-centric health monitoring. Periodic check-ups, while essential, provide only a brief snapshot of an individual's health, often missing the subtle, intermittent signs of developing issues. There is a growing and urgent need for solutions that enable continuous monitoring of physiological parameters in a person's daily environment, empowering individuals to take control of their well-being and facilitating early detection of potential health risks.

The advent of the Internet of Things (IoT) has unlocked unprecedented possibilities in this domain. Low-cost microcontrollers, miniaturized sensors, and ubiquitous wireless connectivity have converged to create the "Internet of Medical Things" (IoMT), a network of interconnected devices that can collect, transmit, and analyze health data in real-time. These technologies have the potential to democratize healthcare, making continuous monitoring accessible and affordable for a broader population.

However, many existing consumer-grade wearable devices are limited in scope. While they can track metrics like heart rate and step count, they often lack the capability to measure or infer more complex clinical parameters like blood pressure, which remains a cornerstone of cardiovascular health assessment. The standard method for blood pressure measurement—the cuff-based sphygmomanometer—is obtrusive, inconvenient for continuous use, and provides only sporadic readings. Our project is motivated by the desire to bridge this gap, leveraging the rich information contained within the photoplethysmography (PPG) signal—a simple optical measurement—to unlock deeper insights into a user's health state non-invasively.

1.2. Problem Statement

The core problem our project seeks to address is the design and development of a low-cost, non-invasive, and continuous health monitoring system capable of not only measuring fundamental vital signs but also accurately predicting advanced health metrics, such as blood pressure and stress levels, using machine learning.

This problem encompasses several sub-challenges:

1. **Hardware Integration:** Selecting and integrating appropriate sensors and a microcontroller to create a reliable data acquisition device.
2. **Data Transmission:** Establishing a robust and reliable communication channel between a portable edge device and a backend processing server, especially when the server may be on a private network without a static public IP address.
3. **Machine Learning Complexity:** Developing and training machine learning models that can infer complex parameters like blood pressure from a simple, single-source PPG signal, a non-trivial task requiring sophisticated signal processing and feature engineering.
4. **Data Standardization:** Devising a signal processing pipeline to ensure that data from a low-cost sensor used in deployment is comparable to the high-fidelity clinical data used for model training.

5. **System-Level Integration:** Ensuring all components—hardware, firmware, communication pipeline, backend server, and user interface—work together seamlessly to provide a cohesive and user-friendly experience.

1.3. Aims and Objectives

The primary aim of our project is to create a fully functional proof-of-concept prototype of the "PenthuAura" health monitoring system. To achieve this aim, our team established the following specific objectives:

- **Objective 1: Design and build a portable data acquisition device.** This involves selecting an ESP32 microcontroller, a MAX30102 PPG/SpO2 sensor, and a MAX30205 temperature sensor, and integrating them into a working circuit with a local display and user input controls.
- **Objective 2: Develop robust firmware for the edge device.** The firmware must manage the measurement cycle, handle sensor readings, buffer and transmit data in chunks, and manage power consumption through sleep modes.
- **Objective 3: Develop a sophisticated machine learning pipeline.** This includes developing a signal standardization process, extracting features, and training and tuning multiple models (Random Forest and Isolation Forest) to predict blood pressure, stress, anomalies, and overall health state.
- **Objective 4: Implement a reliable communication architecture.** This objective focuses on using ThingSpeak and Ngrok to create a resilient data pipeline between the ESP32 and a Raspberry Pi server.
- **Objective 5: Create an intuitive user interface.** This includes displaying real-time results and alerts on the device's LCD screen and developing a comprehensive web-based dashboard for configuration and viewing historical data.
- **Objective 6: Test and validate the end-to-end system.** This involves verifying that the entire system functions as designed, from data acquisition to the final presentation of results.

1.4. Scope and Limitations

This project is an engineering proof-of-concept and an academic exploration into the capabilities of modern IoT and machine learning technologies in healthcare. It is essential to define its scope and acknowledge its limitations:

- **Scope**:
    - The project covers the complete end-to-end design, from hardware schematics to a fully operational backend server.
    - It focuses on using the PPG signal as the primary source for advanced metric prediction.

- It includes the development of a user-friendly web interface for device management.
- The system is designed for use by a single user at a time in a stationary or semi-stationary state to ensure signal quality.

- **Limitations**:
  - **Not a Medical Device:** The PenthuAura system is a prototype and is not a certified medical device. Its readings and predictions are for informational purposes only and should not be used for clinical diagnosis or treatment.
  - **Model Generalization:** The machine learning models are trained on the VitalDB dataset. While extensive, their performance may vary with individuals whose physiological characteristics differ significantly from the training population.
  - **Motion Artifacts:** Like all PPG-based systems, the signal quality is susceptible to motion. The current version does not implement advanced motion artifact removal algorithms.
  - **Power Consumption:** While power-saving modes are implemented, a detailed analysis and optimization of battery life were not primary objectives of this project.

1.5. Team Contribution

This project was the result of a dedicated and collaborative team effort. The responsibilities were distributed among the members to leverage individual strengths and ensure comprehensive coverage of all project domains. The primary contributions were as follows:

- **Diaa Ahmed Hussien: Project Lead & Machine Learning Engineer**
  - Oversaw the overall project timeline and integration of different modules.
  - Led the development of the machine learning pipeline, including data pre-processing from the VitalDB, feature engineering, and the training and tuning of the final Random Forest and Isolation Forest models.
- **Yehia Mohamed Mahmoud: Embedded Systems Engineer**
  - Led the design and development of the ESP32 firmware (PenthuAura_Health_Monitor.ino).
  - Responsible for hardware selection, circuit design, and implementation of the state machine, sensor interfacing, and power management features.
- **Mohanad Tarek Saad: Backend & Cloud Engineer**
  - Developed and deployed the backend server on the Raspberry Pi.
  - Engineered the communication pipeline using Flask, Ngrok, and ThingSpeak.
  - Configured the systemd service for high availability of the server.
- **Abdulrahman Nageh Abdulrahman: Frontend Developer & System Testing Lead**

- Designed and coded the web-based dashboard (index.html, style.css).
- Led the system-wide testing and validation process, ensuring seamless end-to-end functionality and identifying bugs for resolution.

1.6. Thesis Organization

This thesis is structured into six chapters to logically present the development of the PenthuAura project.

- **Chapter 1:** provides an introduction, outlines the motivation, problem statement, objectives, and scope, and details the team contributions.
- **Chapter 2:** presents a comprehensive literature review of the underlying principles and technologies.
- **Chapter 3:** details the complete system design and methodology, covering the hardware, firmware, backend server, and communication architecture.
- **Chapter 4:** focuses exclusively on the machine learning pipeline, detailing the journey from raw data pre-processing to the final training and tuning of the prediction models.
- **Chapter 5:** describes the implementation and testing of the system, presents the results, and discusses the key challenges faced and the solutions engineered to overcome them.
- **Chapter 6:** concludes the thesis by summarizing the project's achievements and suggests potential directions for future work and enhancement.

Chapter (2)

Background Study and Literature Review

Chapter 2 : Background Study and Literature Review

2.1. Principles of Physiological Signal Monitoring

2.1.1. Photoplethysmography (PPG)

Photoplethysmography is a non-invasive optical technique used to detect volumetric changes in blood in the peripheral circulation. The term itself is derived from the Greek words 'photo' (light), 'plethysmos' (increase), and 'graphein' (to write). It operates on the principle that blood absorbs more light than the surrounding tissue. By shining a light-emitting diode (LED) onto the skin and measuring the amount of light reflected or transmitted by a photodetector, one can visualize the pulsatile nature of arterial blood flow.
The PPG waveform contains two main components:

- **AC Component:** This is the pulsatile component synchronized with the cardiac cycle. It represents the changes in blood volume between the systolic and diastolic phases. The frequency of the AC component corresponds directly to the Heart Rate (HR).
- **DC Component:** This is the non-pulsatile component that relates to the bulk absorption of light by skin tissue, bone, and non-pulsating arterial and venous blood.

From these components, two of the most common vital signs can be derived:

- **Heart Rate (HR):** Calculated by measuring the time interval between consecutive peaks in the AC component.
- **Oxygen Saturation (SpO2):** This requires using two wavelengths of light, typically red (around 660 nm) and infrared (around 940 nm). The principle is based on the fact that oxygenated hemoglobin (HbO2) absorbs more infrared light and allows more red light to pass through, while1 deoxygenated hemoglobin (Hb) absorbs more red light and allows more infrared light to pass through. By measuring the ratio of the AC to DC components for both red and infrared light, the ratio of oxygenated to deoxygenated blood can be estimated.

2.1.2. Non-Invasive Body Temperature Sensing

Body temperature is a fundamental vital sign indicating the body's metabolic state. For wearable devices, skin temperature is the most practical measurement. Contact temperature sensors, such as the MAX30205 used in our project, are ideal. The MAX30205 is a clinical-grade temperature sensor that provides high accuracy (±0.1°C) and communicates via the I2C protocol.

2.1.3. Techniques for Non-Invasive Blood Pressure (NIBP) Estimation from PPG

Estimating blood pressure (BP) without a cuff is a significant area of research. The morphology of the PPG waveform contains a wealth of information related to cardiovascular dynamics.

- **Pulse Transit Time (PTT):** This established method measures the time delay between an electrocardiogram (ECG) R-peak and a characteristic point on the PPG waveform. It requires two sensors, increasing complexity.
- **PPG Waveform Feature Analysis:** This approach, used in our project, relies solely on features extracted from the PPG signal itself. The shape of the PPG wave—its amplitude, width, and characteristic points—changes in response to variations in blood pressure. Machine learning models can be trained to learn this complex relationship.

2.1.4. Heart Rate Variability (HRV) and Stress Analysis

Heart Rate Variability (HRV) is the variation in the time interval between consecutive heartbeats. A high HRV is generally a sign of a healthy, adaptable autonomic nervous system (ANS), while a low HRV is often an indicator of stress. The ANS has two branches: the sympathetic (fight-or-flight) and parasympathetic (rest-and-digest). HRV analysis provides a window into the balance between these two branches. Features extracted from HRV can be categorized into:.

- **Time-Domain Features:** Statistics calculated from the R-R intervals (e.g., SDNN, RMSSD).
- **Frequency-Domain Features:** Analysis of the power spectral density (PSD) of the R-R interval series.
- **Non-Linear Features:** Advanced metrics like Detrended Fluctuation Analysis (DFA) that capture the complexity of the heart rate signal. These are powerful predictors of stress.

2.2. The Internet of Medical Things (IoMT)

2.2.1. System Architectures: Edge, Fog, and Cloud

- **Cloud Computing:** A centralized approach where all raw sensor data is sent to powerful cloud servers. This offers immense computational power but can suffer from high latency and privacy concerns.
- **Edge Computing:** Data processing occurs on or near the sensor device itself. This minimizes latency and enhances data privacy but is limited by the device's computational power.
- **Fog/Hybrid Computing:** This approach, which most closely mirrors the PenthuAura architecture, is a balance between the two. The edge device (ESP32) performs data acquisition. This data is then sent to a local gateway (the Raspberry Pi) for more intensive processing. This architecture optimizes for both responsiveness and computational power.

2.2.2. Communication Platforms and Protocols in Healthcare

- **Protocols:** Standard protocols like Wi-Fi and Bluetooth Low Energy (BLE) are common. For sending data to the cloud, application-layer protocols like MQTT (Message Queuing Telemetry Transport) and HTTP (Hypertext Transfer Protocol) are widely used. We chose HTTP for its ubiquity and simplicity in interacting with our chosen cloud services.
- **Platforms:** Cloud platforms like AWS IoT, Google Cloud IoT, and Microsoft Azure provide comprehensive services. For rapid prototyping, platforms like **ThingSpeak** are invaluable. ThingSpeak offers a simple, RESTful API-based service to log and retrieve data, making it an excellent message broker.

2.3. Machine Learning in Physiological Signal Analysis

2.3.1. Time-Series Feature Extraction (NeuroKit2)

Raw physiological signals like PPG are high-dimensional and noisy. To make them suitable for machine learning, meaningful features must be extracted. **NeuroKit2** is a Python library specifically designed for physiological signal processing. It provides validated algorithms to clean signals, detect characteristic points (like PPG peaks), and compute a wide array of clinically relevant HRV and PPG analysis features.

2.3.2. Supervised Learning for Prediction

Supervised learning involves training a model on a labeled dataset.

- **Regression with Random Forest Regressor:** For predicting blood pressure, we selected the Random Forest Regressor. A Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees at training time. The final prediction is the average of the predictions from all the individual trees. This method is highly effective because it corrects for the habit of individual decision trees to overfit to their training set.
- **Classification with Random Forest Classifier:** For determining Health State and Stress level, we employed the Random Forest Classifier. It builds a forest of decision trees, and the final decision is made by a majority vote. Its robustness against overfitting and its ability to handle complex feature interactions made it an ideal choice.

2.3.3. Unsupervised Learning for Anomaly Detection

Unsupervised learning is used when the data is not labeled.

- **Isolation Forest:** For our anomaly detection module, we chose the Isolation Forest algorithm. It operates on the principle that anomalies are "few and different," making them

easier to isolate than normal data points. It builds an ensemble of "Isolation Trees" where anomalous points will have a shorter average path length.

2.4. Review of Key Enabling Technologies

- **Microcontrollers in IoT: The ESP32:** The ESP32 is a powerful, low-cost microcontroller that has become a staple of IoT projects due to its dual-core processor, integrated Wi-Fi and Bluetooth, and support for low-power sleep modes.
- **Sensor Technology: MAX30102 and MAX30205:** The MAX30102 is an integrated pulse oximetry and heart-rate monitor module. The MAX30205 is a high-accuracy digital temperature sensor. Both use the I2C communication protocol, allowing for simple integration with the ESP32.

2.5. Identification of the Research Gap

While many projects have explored cuffless blood pressure estimation, our literature review identified a gap in the creation of a complete, end-to-end, open-source system that is both low-cost and highly functional. The PenthuAura project was conceived to fill this gap by integrating multiple advanced predictions, engineering a novel communication pipeline, and providing a detailed methodology, including a critical data standardization step.

Chapter (3)

System Design and Methodology

Chapter 3 : System Design and Methodology

3.1. High-Level System Architecture

The PenthuAura system is designed with a distributed, multi-layered architecture to balance processing load, power consumption, and responsiveness. The architecture, as depicted in Figure 3.1, can be broken down into three primary layers: the Edge Layer, the Communication & Cloud Layer, and the Backend Processing Layer.

1. **Edge Layer (PenthuAura Device):** This layer consists of the physical device. Its responsibilities are data acquisition, user interaction, and data transmission. It is built around an ESP32 microcontroller and integrates the MAX30102 and MAX30205 sensors.
2. **Communication & Cloud Layer (ThingSpeak & Ngrok):** This layer acts as the intelligent conduit between the edge and backend. **ThingSpeak** is used as a message broker for the RPi to post its public Ngrok URL and the final results. **Ngrok** creates a secure public URL that tunnels to the Flask web server running on the Raspberry Pi, solving the problem of dynamic IP addresses.
3. **Backend Processing Layer (Raspberry Pi Server):** This is the computational core. A Raspberry Pi runs a Python script that hosts a Flask web server. Upon receiving the raw PPG data, it executes the entire machine learning pipeline.

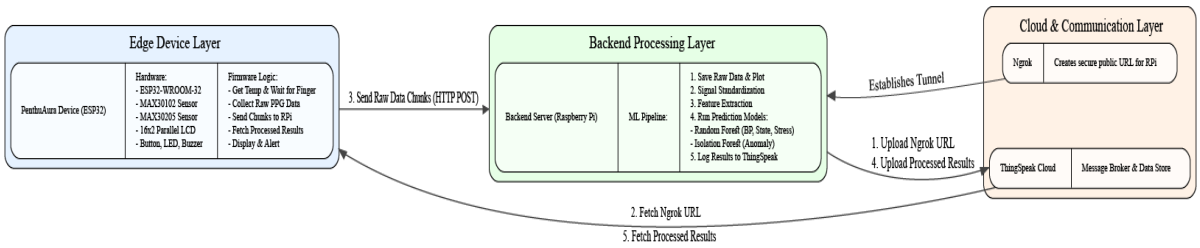

Figure 3.1: PenthuAura High-Level System Architecture

3.2. Hardware Design

3.2.1. Component Selection and Justification

- **Microcontroller (ESP32-WROOM-32):** Chosen for its powerful dual-core processor and integrated Wi-Fi.
- **PPG/SpO2 Sensor (MAX30102):** Selected for its integration of red and IR LEDs and processing circuitry.
- **Temperature Sensor (MAX30205):** A clinical-grade sensor chosen for its high accuracy.
- **Display (16x2 Parallel LCD):** A standard 16x2 character LCD was chosen for its low cost and wide availability. It was connected in 4-bit parallel mode, which offers a balance between the number of GPIO pins required (6) and the speed of data transfer.
- **User Input (Push Button):** A single tactile push button provides a versatile user interface.
- **Alerts (RGB LED and Buzzer):** A common-cathode RGB LED and an active buzzer provide multi-modal alerts.

3.2.2. Circuit Design and Schematic

The circuit was designed for functionality and assembled on a standard dot-matrix PCB (perfboard). The ESP32 is the central component, powered directly via its **5V VIN pin** from a standard USB power source. The MAX30102 and MAX30205 sensors are connected to the ESP32's primary I2C bus. The 16x2 LCD was connected in 4-bit mode, requiring a total of 6 GPIO pins (RS, E, D4, D5, D6, D7).

3.2.3. Prototype Assembly

The initial prototype was assembled on a breadboard to validate the circuit design. After successful validation, the components were soldered onto a dot-matrix perfboard to create a more permanent prototype. Wires were used to establish all connections as per the circuit schematic.

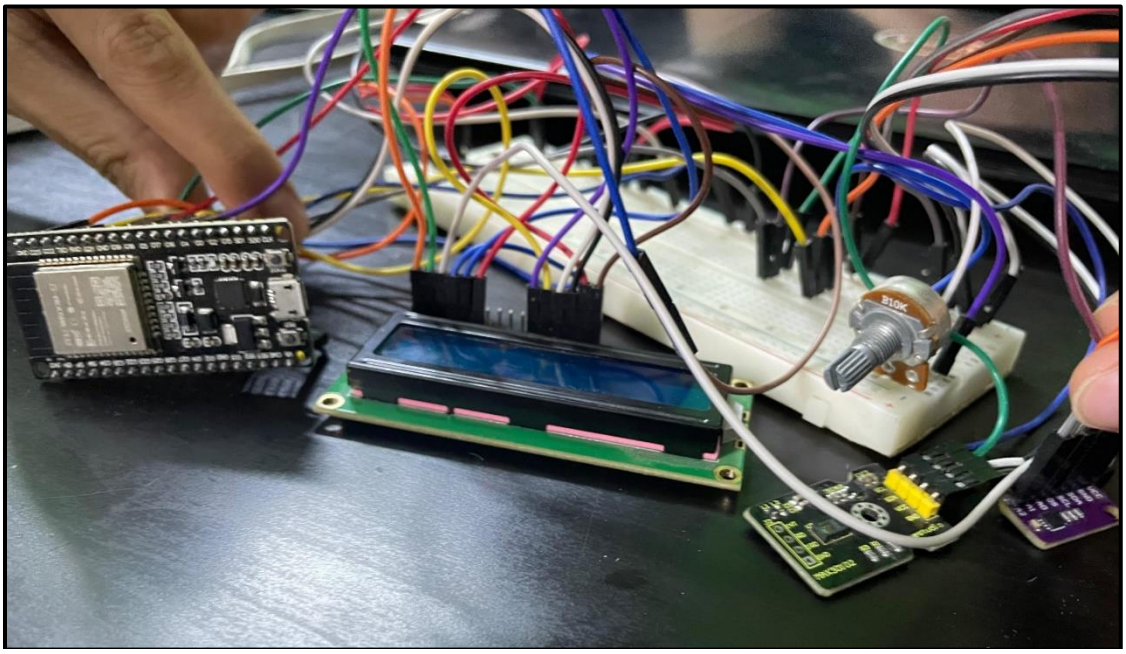

Figure 3.2: Final Assembled Prototype on Breadboard

3.3. ESP32 Firmware Design

3.3.1. The Measurement Cycle State Machine

The firmware's logic is governed by a robust state machine implemented within the main loop. This ensures a predictable and orderly execution of the measurement process. The CycleState enum defines the possible states, and a central runMeasurementCycle() function acts as the state transition engine.

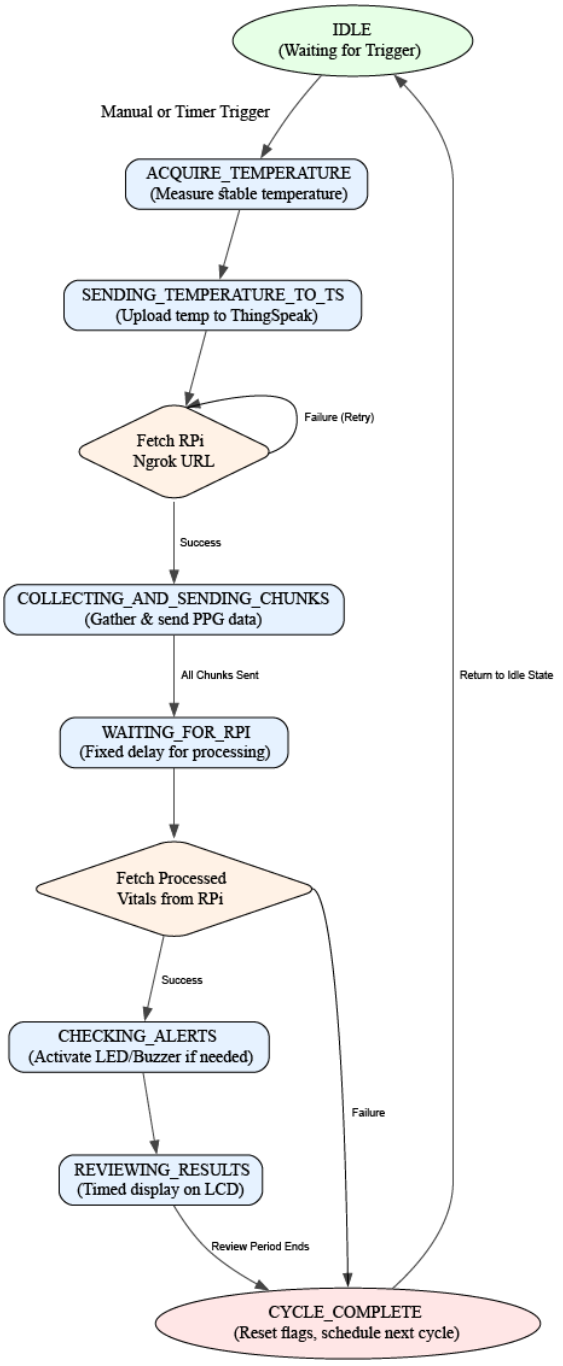

Figure 3.3: Firmware State Machine Flowchart

3.3.2. Data Acquisition and Chunking

To manage the large volume of data generated by the PPG sensor (100 samples/second), a data chunking strategy was implemented. The firmware collects data for a full 10 seconds (1000 samples) into two arrays (irChunkBuffer and redChunkBuffer). Once a chunk is full, the firmware sends it to the Raspberry Pi server in a single HTTP POST request. This process is repeated three times to gather 30 seconds of data. A key feature of the firmware is its finger detection logic.

3.3.3. Dynamic Service Discovery via ThingSpeak

A significant challenge in IoT systems is for an edge device to locate its backend server. Our firmware solves this with a two-step service discovery process:

1. The ESP32 makes an HTTP GET request to a ThingSpeak channel field where the Raspberry Pi posts its public URL.
2. The firmware parses the response to extract the Ngrok URL and uses this as the base address for all subsequent data chunk submissions.

3.3.4. Power Management and RTC Data Persistence

The firmware leverages the ESP32's sleep modes:

- **Light Sleep:** After a short period of inactivity, the device enters a light sleep mode.
- **Deep Sleep:** After a longer period of inactivity, the device enters deep sleep. The device is configured to wake up either via a press of the user button or when an internal RTC timer expires.

To preserve important data across deep sleep cycles, RTC_DATA_ATTR is used to store variables like the historical data logs in the ESP32's RTC memory.

3.4. Backend Server Design (Raspberry Pi)

3.4.1. Server Environment (Flask, Python)

The backend is a Python script designed to run on a Raspberry Pi. It uses the Flask micro-framework to create a lightweight web server to receive the data from the ESP32.

3.4.2. ML Pipeline and Processing Workflow

The core of the server is the /data endpoint. When an HTTP POST request is received:

1. It validates and parses the incoming JSON payload.
2. The data is appended to a list of received chunks.
3. Once three chunks have been received, it triggers the main run_ml_pipeline() function in a separate thread.
4. The pipeline function then executes all the steps defined in Chapter 4.
5. Finally, it formats the results and uploads them to ThingSpeak.

3.4.3. Data Persistence and Visualization

For debugging and analysis, before running the ML pipeline, the server script performs two key actions:

1. **Saves Raw Data:** The combined 30-second PPG signal is saved as a timestamped .csv file.
2. **Saves Waveform Plot:** A plot of the IR signal waveform is generated using Matplotlib and saved as a timestamped .png image.

3.4.4. Automated Service Deployment (systemd)

To ensure the backend server is robust and runs automatically, a systemd service was created. systemd is the standard service manager in most Linux distributions. A service unit file was written that defines how to start the Python Flask script and configured it to automatically restart the script if it ever crashes.

3.5. User Interface Design (LCD and Web Dashboard)

A dual-interface approach was taken.

- **LCD Interface:** The 16x2 character LCD provides at-a-glance information. A single button allows the user to cycle through five different screens: (1) Main Vitals, (2) WiFi Status, (3) Alert Status, (4) Server Connection Status, and (5) System Info.
- **Web Dashboard:** The ESP32 also hosts a sophisticated web dashboard, built with HTML, CSS, and JavaScript. It allows users to view live data, configure Wi-Fi, and view historical measurements.

Chapter (4)

Machine Learning Pipeline: From Data to Deployment

Chapter 4 : Machine Learning Pipeline: From Data to Deployment

4.1. Dataset Acquisition and Labeling

4.1.1. Introduction to the VitalDB Clinical Dataset

The foundation of our supervised machine learning project is the **VitalDB dataset**, a public database containing high-resolution physiological data from surgical patients. This dataset is invaluable as it provides time-synchronized waveforms for PPG (PLETH signal), ECG, and Invasive Arterial Blood Pressure (ABP), which serves as the ground truth for our blood pressure regression model.

4.1.2. Manual Data Labeling for Supervised Learning

A significant challenge is that VitalDB lacks high-level labels for complex states like "stress." To create a viable dataset for our classification models, our team undertook a meticulous labeling process. Based on an extensive review of medical literature correlating Heart Rate Variability (HRV) metrics with stress and patient acuity, we developed a set of rules and thresholds. These rules were used to programmatically apply "Stress" and "Health State" (Normal, Moderate, Critical) labels to each 30-second segment of the VitalDB data, transforming it into a fully-labeled dataset.

4.2. Signal Standardization and Pre-processing

For our machine learning models to be effective, it was essential that the data they were trained on (from VitalDB) and the data they would see in live prediction (from the MAX30102) were in a comparable format

4.2.1. The Three-Stage Standardization Process

This process, applied to both the VitalDB PLETH signal and the live MAX30102 IR signal, involved three key stages:

1. **Baseline Correction:** The raw signals from both sources exhibit a large DC offset. The first step is to remove this baseline wander by subtracting the mean of the signal window.
2. **Bandpass Filtering:** To eliminate noise, we applied a digital Butterworth bandpass filter to allow frequencies between 0.5 Hz and 4.0 Hz (corresponding to 30-240 beats per minute) to pass through.

4.2.2. Verification of Standardization

Figure 4.1 visually demonstrates the success of this process. It shows the raw MAX30102 signal, its processed version, and the similarly processed VitalDB PLETH signal. The two processed signals are now highly comparable in form.

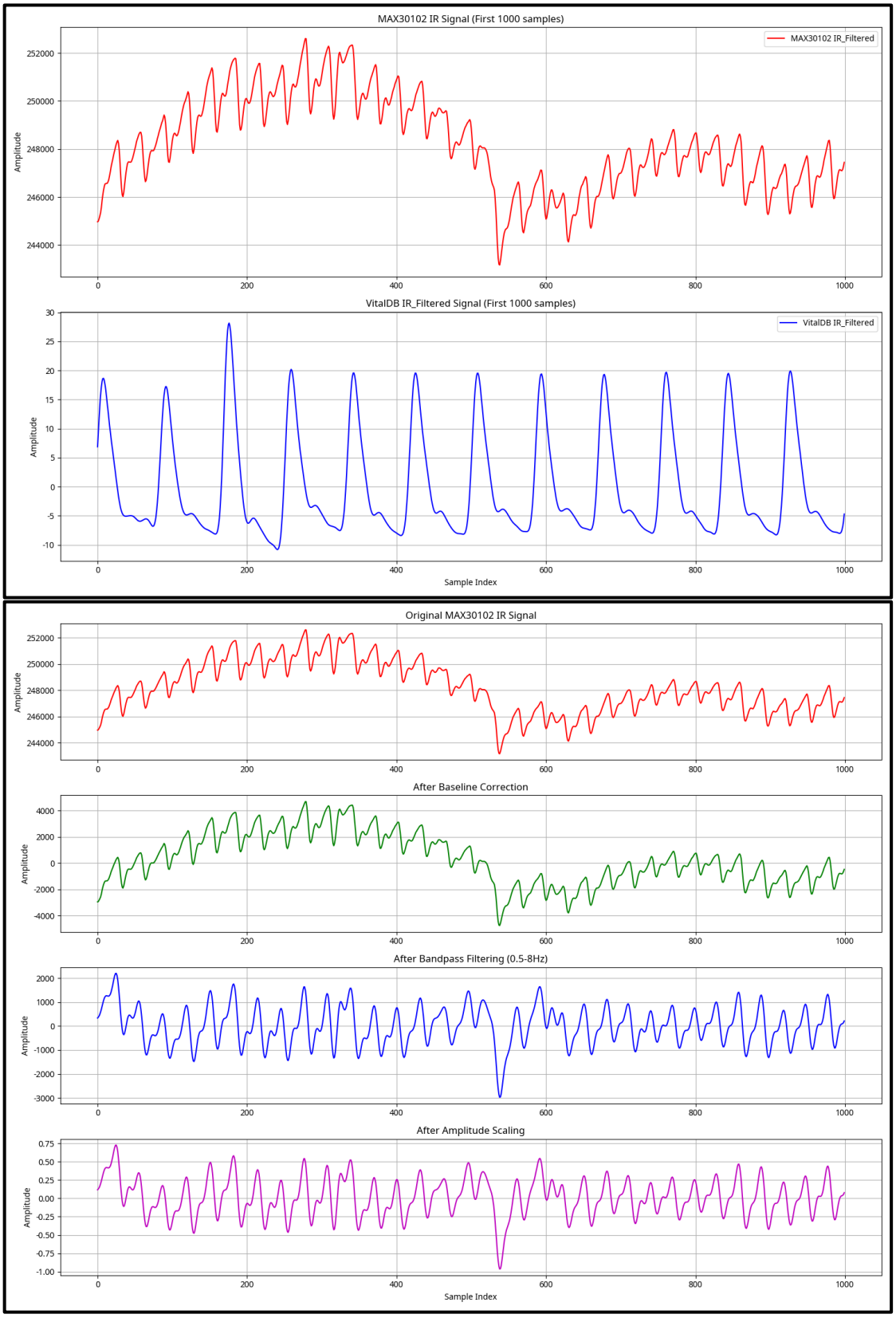

Figure 4.1: Signal Standardization Pipeline

4.3. Feature Engineering and Selection

4.3.1. Feature Extraction using NeuroKit2

For each 30-second segment of the standardized PPG signal, a comprehensive set of features was extracted using the NeuroKit2 library to derive clinically relevant features related to heart rate variability and PPG waveform morphology.

4.3.2. Rationale for TOP_15_FEATURES Selection

From an initial pool of over 50 features, we used feature importance scores from an initial model run to identify a smaller subset of the most informative features. This resulted in the TOP_15_FEATURES list, which includes a mix of SpO2, temperature, heart rate statistics, and key HRV metrics.

4.4. Baseline Model Training with Random Forest

Before attempting to optimize the models, we established a baseline by training untuned versions of our chosen models on an 80/20 split of our processed dataset. The baseline models performed reasonably well, but the evaluation revealed a key weakness in the Health State model's ability to identify minority classes.

4.5. Model Enhancement and Optimization

4.5.1. Addressing Class Imbalance with SMOTE

To address the poor performance on critical health states, we employed the SMOTE (Synthetic Minority Over-sampling Technique) algorithm. SMOTE works by creating new, synthetic data points for the minority classes, balancing the class distribution in the training set and allowing the model to learn the characteristics of the under-represented classes more effectively.

4.5.2. Hyperparameter Tuning using RandomizedSearchCV

To further improve performance, we tuned the hyperparameters of the Random Forest models for both blood pressure and health state using RandomizedSearchCV, which efficiently samples parameter settings to find an optimal combination.

4.6. Final Model Evaluation and Deployment

The impact of our enhancements was significant, with the final tuned models showing marked improvement in prediction accuracy, especially for the critical health states. Once training and evaluation were complete, the final, optimized models and the feature scaler were serialized and saved using joblib to be loaded by the Raspberry Pi server for live predictions.

Chapter (5)

Implementation, Testing, and Results

Chapter 5 : Implementation, Testing, and Results

5.1. System Implementation Overview

The implementation phase involved integrating the hardware prototype, the ESP32 firmware, the backend server script, and the web dashboard into a single, cohesive system. The final Raspberry Pi server script was configured to load the tuned models. The ESP32 was flashed with the final version of the firmware. The systemd service was enabled on the Raspberry Pi to ensure the backend server would start on boot.

5.2. Testing, Validation, and Performance

5.2.1. Unit Testing

- **Sensors:** The accuracy of the MAX30102 and MAX30205 was verified against a commercial pulse oximeter and thermometer.
- **Firmware Modules:** Functions for WiFi connection, data chunking, and LCD display were tested individually.
- **Backend:** The Flask server was tested locally using tools like curl to send mock JSON data and verify the response.

5.2.2. Integration Testing (End-to-End Data Flow)

This was the most critical testing phase. A full measurement cycle was initiated, and we monitored the entire data flow in real-time to confirm that the ESP32 successfully fetched the Ngrok URL, sent the data chunks, and that the Raspberry Pi received them, ran the ML pipeline, and uploaded the results to ThingSpeak for the ESP32 to fetch and display.

5.2.3. User Interface Testing

The functionality of both the LCD menu and the web dashboard was thoroughly tested, including cycling through all screens, saving new Wi-Fi configurations, and verifying the real-time update of the history table.

5.3. Analysis of Live Results

5.3.1. Machine Learning Model Performance

The final performance of our models, evaluated on the held-out test set from VitalDB, is summarized below.

Table 5.1: Blood Pressure Model Performance

Metric	Systolic BP (SBP)	Diastolic BP (DBP)
Mean Absolute Error (MAE)	7.944 mmHg	7.944 mmHg
Mean Squared Error (MSE)	192.978	192.978

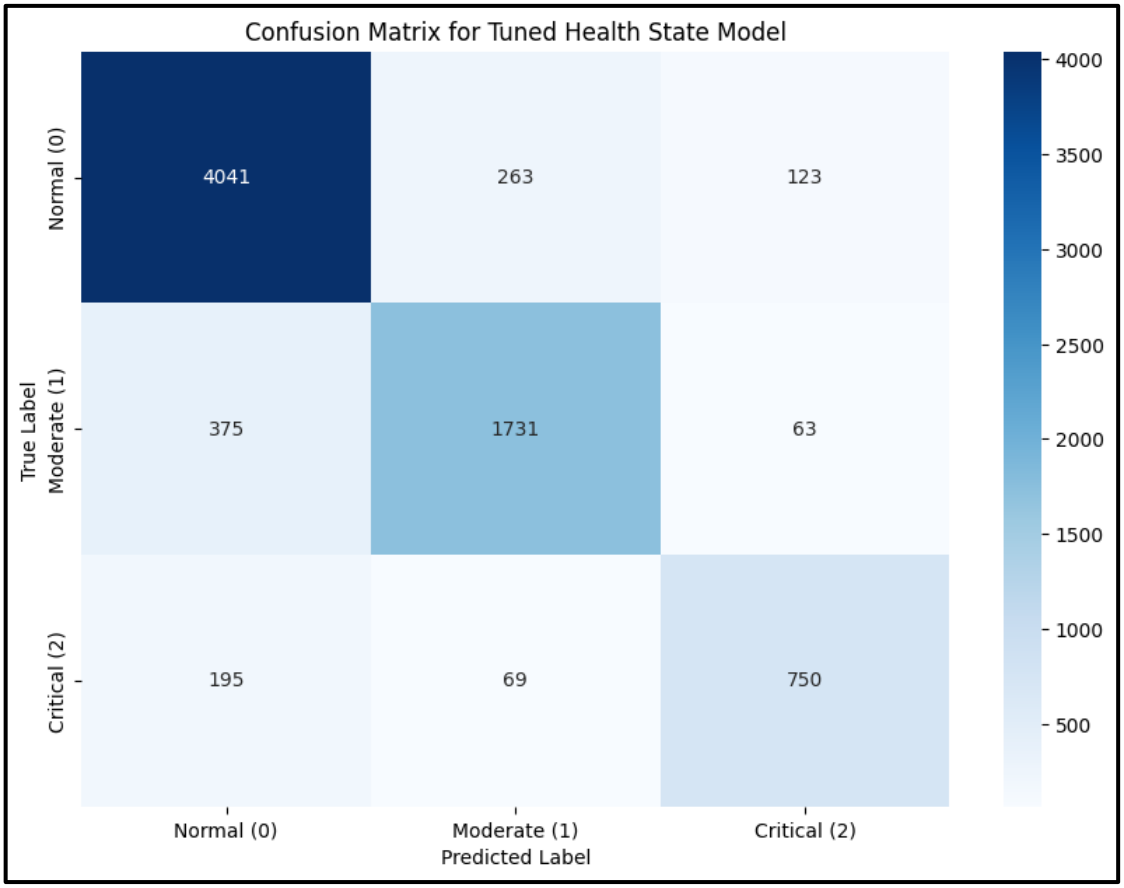

Figure 5.1: Health State Model Confusion Matrix - Tuned

Table 5.2: Health State Model Classification Report - Tuned

Class	Precision	Recall	F1-Score
Normal	0.88	0.91	0.89
Moderate	0.84	0.80	0.82
Critical	0.80	0.74	0.77
Accuracy			0.86

As the results show, the final tuned models achieved a high degree of accuracy. The use of SMOTE was particularly effective, significantly boosting the recall for the "Critical" state.

5.3.2. Sample Output Analysis

During live testing, the backend server generated output files for each measurement. Figure 5.2 shows a sample waveform plot, and Table 5.3 shows a snippet from the corresponding raw data CSV file.

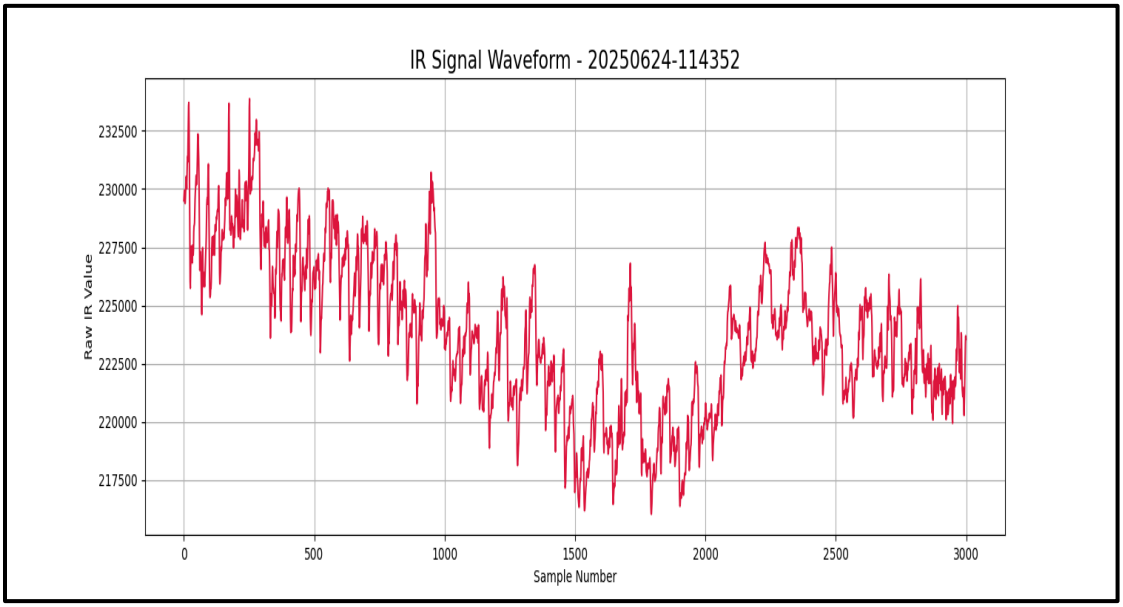

Figure 5.2: Sample IR Waveform from a Live Test

Table 5.3: Snippet of Saved Raw Data from a Live Test (raw_data_...csv)

IR_Readings out of MAX30102	Red_Readings out of MAX30102
227262	186041
227177	186185
228016	186431
228449	186566
228463	186535
228193	186422
228251	186541
228539	186644
228811	186694
228715	186678

5.3.3. Screenshots of UI in Operation

Figure 5.3: LCD Display Showing Final Results

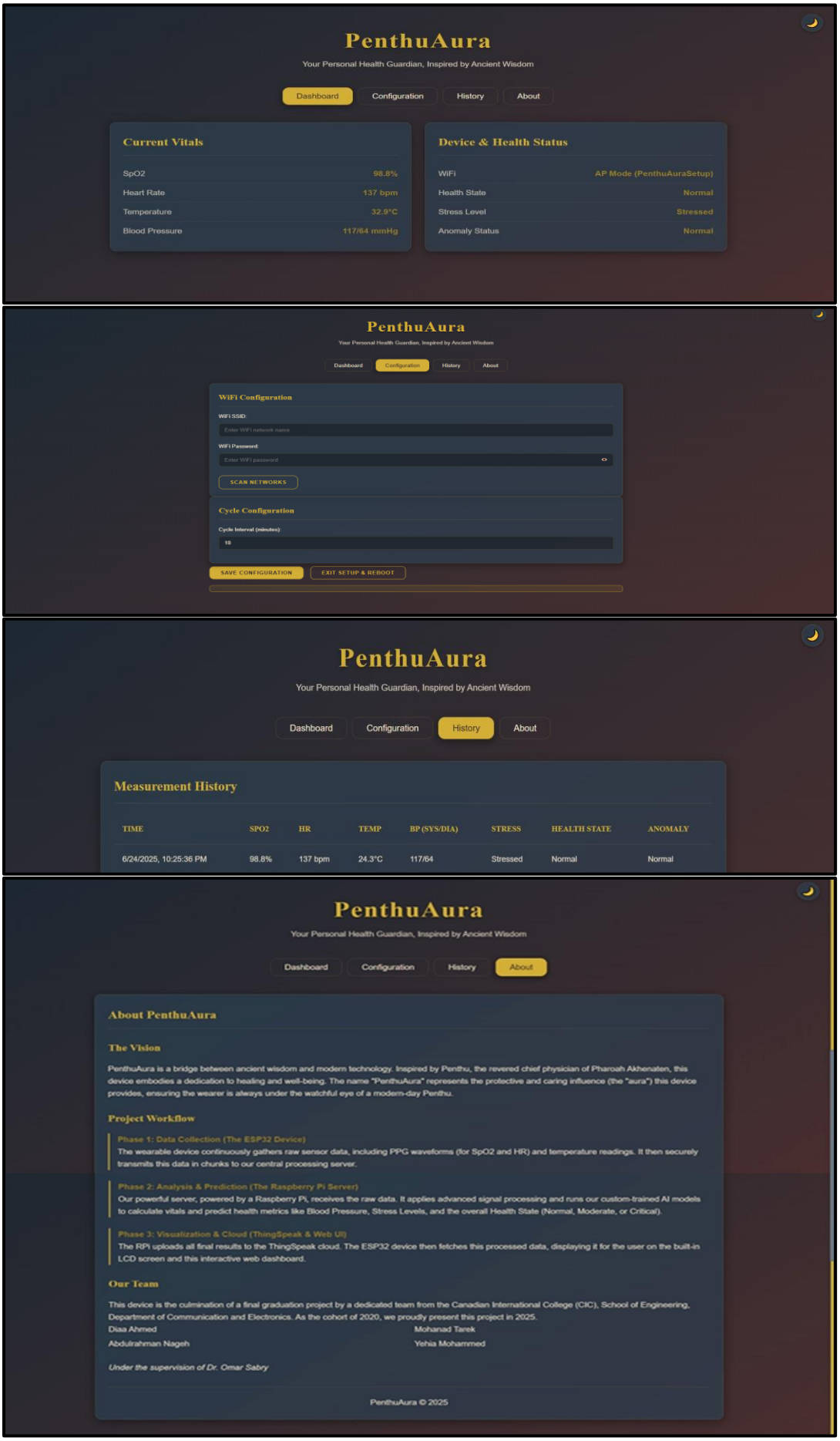

Figure 5.4: Web Dashboard Displaying Real-time Vitals

5.4. Project Challenges and Resolutions

5.4.1. Hardware Challenge: Mitigating Noise in PPG Sensor Readings

- **Challenge:** Initial readings from the MAX30102 sensor were noisy and inconsistent. Without an enclosure, the sensor was highly susceptible to ambient light interference and signal degradation from finger movements.
- **Resolution:** This was addressed through software processing and user procedure. The primary solution was the implementation of the **bandpass filter** in the backend processing pipeline. This digital filter proved highly effective at removing out-of-band noise. Procedurally, the user is instructed to cup their hand over the sensor during measurement to create a darker environment.

5.4.2. Networking Challenge: Establishing a Robust ESP32-to-RPi Connection

- **Challenge:** The core problem was enabling the ESP32 to reliably send data to the Raspberry Pi server on a private network.
- **Resolution:** The team devised a cloud-brokering solution using **Ngrok** to create a persistent public URL for our local Raspberry Pi server and **ThingSpeak** for the ESP32 to read this URL. This decoupled architecture proved to be exceptionally robust.

5.4.3. Firmware Challenge: Managing ESP32 Memory Constraints

- **Challenge:** The ESP32 has limited RAM. Storing the large JSON libraries, the web server code, and the data buffers simultaneously led to instability.
- **Resolution:** We meticulously optimized the firmware's memory usage by using the ArduinoJSON library for precise memory allocation and ensuring that large buffers were only allocated when needed.

5.4.4. Machine Learning Challenge: The Necessity of SMOTE for Accurate Classification

- **Challenge:** Our initial Health State model was very poor at correctly identifying the "Critical" state due to class imbalance in the dataset.
- **Resolution:** The machine learning lead implemented the **SMOTE** technique to generate synthetic data for the minority classes. The model retrained on this new balanced data showed a dramatic improvement in recall for the critical class.

5.4.5. Deployment Challenge: Ensuring High Availability of the Backend Server

- **Challenge:** Manually starting the Python server script on the Raspberry Pi after every reboot was impractical and unreliable.
- **Resolution:** The backend lead implemented a professional deployment solution using **systemd**, the native service manager in Raspbian OS, to automatically launch the server script on system boot and restart it if it ever crashed.

Chapter (6)

Conclusion and Future Work

Chapter 6 : Conclusion and Future Work

6.1. Conclusion

This project successfully achieved its primary aim: the design, implementation, and validation of "PenthuAura," a comprehensive, low-cost IoT health monitoring system. Through a collaborative team effort, we have produced a fully functional proof-of-concept that demonstrates the power of combining modern embedded systems, cloud communication strategies, and machine learning to create accessible and intelligent health technology.

Our team successfully engineered an end-to-end pipeline, from a custom-built ESP32 device to a Raspberry Pi backend that performs complex predictions. The novel use of ThingSpeak and Ngrok as a communication bridge proved to be a highly effective solution. The core contribution of this work lies in the sophisticated machine learning pipeline, which not only predicts standard vitals but also infers advanced metrics like cuffless blood pressure and stress levels. The meticulous standardization of the sensor and clinical data and the use of advanced techniques like SMOTE to create a fair and effective classification model are significant technical achievements.

6.2. Future Work

6.2.1. Hardware Enhancements

- **Custom PCB and Ergonomic Enclosure:** A top priority for future work is to design a custom PCB to miniaturize the device and improve its reliability. Concurrently, designing and **3D-printing a custom, ergonomic enclosure** is critical. A proper enclosure would protect the electronics and provide a snug, dark chamber for the sensor, dramatically improving signal quality.
- **Battery Integration and Power Optimization:** Incorporating a rechargeable Li-Po battery and a dedicated power management IC would make the device truly portable.

6.2.2. Firmware and Software Improvements

- **Over-the-Air (OTA) Updates:** Implementing an OTA update mechanism would allow for remote firmware upgrades.
- **Enhanced Security:** End-to-end encryption between the ESP32 and the server could be implemented for an additional layer of data security.

6.2.3. Machine Learning Advancements

- **Deep Learning Models:** Exploring the use of deep learning models, such as LSTMs or CNNs, could potentially improve prediction accuracy.
- **Personalized Models:** A future version could implement an online learning mechanism, where the models fine-tune themselves to an individual user over time.

6.2.4. Path towards a Medical-Grade Device

The ultimate goal for a system like PenthuAura would be to achieve clinical validation. This would involve conducting clinical trials to compare the device's predictions against gold-standard medical equipment and adhering to regulatory standards for medical devices.

Chapter (7)

References

Chapter 7 : References

1. Allen, J., 2007, “Photoplethysmography and its application in clinical physiological measurement”, Physiological Measurement, Vol. 28, No. 3, pp. R1.
2. Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology, 1996, “Heart rate variability: standards of measurement, physiological interpretation, and clinical use”, Circulation, Vol. 93, No. 5, pp. 1043-1065.
3. Breiman, L., 2001, “Random Forests”, Machine Learning, Vol. 45, No. 1, pp. 5-32.
4. Liu, F. T., Ting, K. M., and Zhou, Z. H., 2008, “Isolation Forest”, Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, pp. 413-422.
5. Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P., 2002, “SMOTE: Synthetic Minority Over-sampling Technique”, Journal of Artificial Intelligence Research, Vol. 16, pp. 321-357.
6. Elgendi, M., 2013, “On the analysis of fingertip photoplethysmogram signals”, Biomedical Signal Processing and Control, Vol. 8, No. 6, pp. 532-541.
7. "ThingSpeak IoT Analytics," The MathWorks, Inc. [Online]. Available: https://thingspeak.com. [Accessed: Jun. 24, 2025].
8. "Ngrok - Secure Tunnels to Localhost," Ngrok, Inc. [Online]. Available: https://ngrok.com. [Accessed: Jun. 24, 2025].

Chapter (8)

Appendices

Chapter 8 : Appendices

8.1. Appendix A: Source Code

This appendix contains key source code listings for the major components of the PenthuAura project. The complete, fully commented code is provided in the digital submission accompanying this thesis.

8.1.1. ESP32 Firmware (PenthuAura_Health_Monitor.ino)

```
void runMeasurementCycle() {
  if (!isCycleRunning) {
    return;
  }

  switch (currentCycleState) {
    case IDLE:
      isCycleRunning = false;
      break;

    case ACQUIRE_TEMPERATURE:
      Serial.println("Cycle Step 1: Acquiring Temperature...");
      displaySystemMessageOnLCD("Measuring Temp", "Please wait...");
      gCurrentTemp = waitForStableTemperature_impl();
      Serial.printf("Temp Acquired: %.1f C\n", gCurrentTemp);
      currentCycleState = SENDING_TEMPERATURE_TO_TS;
      break;

    case SENDING_TEMPERATURE_TO_TS:
      Serial.println("Cycle Step 2: Sending Temperature to ThingSpeak...");
      displaySystemMessageOnLCD("Sending Temp", "To ThingSpeak...");
      sendTemperatureToThingSpeak();
      currentCycleState = FETCHING_URL_FROM_TS;
      break;

    case FETCHING_URL_FROM_TS:
      Serial.println("Cycle Step 3: Fetching RPi URL...");
      if (fetchNgrokUrlFromThingSpeak()) {
        currentCycleState = COLLECTING_AND_SENDING_CHUNKS;
        chunksSent = 0; chunkSampleCounter = 0; settled = false;
        isFingerSettling = false; collectionStartTime = millis();
      } else {
        // If failed, wait and retry this state in the next loop iteration.
        Serial.println("Failed to fetch URL. Retrying in 5 seconds...");
        displaySystemMessageOnLCD("URL Fetch Fail", "Retrying...");
        delay(5000);
      }
      break;

    case COLLECTING_AND_SENDING_CHUNKS:
      {
        if (!settled) {
```

```
            if (millis() - collectionStartTime >= 5000) {
                settled = true;
                Serial.println("Warm-up complete. Place finger firmly...");
                displaySystemMessageOnLCD("Ready", "Place Finger");
            } else {
                Serial.println("Sensor warming up...");
                displaySystemMessageOnLCD("Sensor Warm-up", "Please Wait...");
                delay(500);
            }
            break;
        }

        long irValue = particleSensor.getIR();
        particleSensor.nextSample();
        if (irValue < FINGER_DETECT_THRESHOLD_IR) {
            isFingerSettling = false;
            break;
        }

        if (!isFingerSettling) {
            isFingerSettling = true;
            fingerSettleStartTime = millis();
            displaySystemMessageOnLCD("Finger Detected", "Settling...");
        }

        if (millis() - fingerSettleStartTime >= FINGER_SETTLE_DURATION_MS) {
            if (chunkSampleCounter == 0 && chunksSent < TOTAL_CHUNKS) {
                displaySystemMessageOnLCD("Collecting...", "Chunk " + String(chunksSent + 1));
            }
            if (chunkSampleCounter < CHUNK_SIZE) {
                // Store IR and Red PPG values in the buffers.
                irChunkBuffer[chunkSampleCounter] = irValue;
                redChunkBuffer[chunkSampleCounter] = particleSensor.getRed();
                chunkSampleCounter++;
            }
            if (chunkSampleCounter >= CHUNK_SIZE) {
                sendDataChunk();
                chunksSent++;
                chunkSampleCounter = 0;
                isFingerSettling = false;
            }
        }
    }
    // Check if all chunks have been sent.
    if (chunksSent >= TOTAL_CHUNKS) {
        Serial.println("All chunks sent to RPi. Waiting for processing...");
        currentCycleState = WAITING_FOR_RPI;
    }
    break;

case WAITING_FOR_RPI:
    Serial.println("Cycle Step 5: Waiting for RPi to calculate and upload...");
    displaySystemMessageOnLCD("Waiting for RPi", "10 seconds...");
    delay(10000);
    currentCycleState = FETCHING_VITALS_FROM_TS;
    break;
```

```
    case FETCHING_VITALS_FROM_TS:
      Serial.println("Cycle Step 6: Fetching All RPi Results...");
      displaySystemMessageOnLCD("Fetching Results", "From ThingSpeak");

      if (fetchAllResultsFromThingSpeak()) {
        addHistoricalData(gCurrentSPO2, gCurrentHR, gCurrentTemp, gSBP, gDBP, gStressStatus,
gAnomalyStatus, gHealthState);
        updateLCD();
        currentCycleState = CHECKING_ALERTS;
      } else {
        Serial.println("Could not fetch all RPi results. Ending cycle.");
        currentCycleState = CYCLE_COMPLETE;
      }
      break;

    case CHECKING_ALERTS:
      Serial.println("Cycle Step 7: Checking for RPi-based Alerts...");

      if (gHealthState == "Critical") {
        displaySystemMessageOnLCD("! CRITICAL !", gAnomalyStatus);
        soundCriticalAlert();
      } else if (gHealthState == "Moderate") {
        displaySystemMessageOnLCD("! MODERATE !", gStressStatus);
        soundModerateAlert();
      } else {
        displaySystemMessageOnLCD("State: Normal", "All systems OK");
      }

      delay(2000);
      currentCycleState = REVIEWING_RESULTS;
      break;

    case REVIEWING_RESULTS:
      if (reviewPeriodStartTime == 0) {
        Serial.println("Cycle Complete. Entering review period...");
        displaySystemMessageOnLCD("Cycle Complete", "Review Results");
        setRgbLed(0, 0, 255);
        reviewPeriodStartTime = millis();
      }

      if (millis() - lastLCDUpdateTime > LCD_UPDATE_INTERVAL) {
        updateLCD();
      }

      if (millis() - reviewPeriodStartTime >= REVIEW_PERIOD_MS) {
        Serial.println("Review period ended.");
        reviewPeriodStartTime = 0;
        currentCycleState = CYCLE_COMPLETE;
      }
      break;

    case CYCLE_COMPLETE:
      Serial.println("Measurement Cycle Complete. Scheduling next cycle and returning to Idle.");

      nextCycleStartTime = millis() + (long)cycle_interval_minutes * 60 * 1000;
```

```
        lastInteractionTime = millis();
        isCycleRunning = false;
        currentCycleState = IDLE;
        break;
    }
}
```

8.1.2. Final Raspberry Pi Server Script

```
def run_ml_pipeline():
    global received_chunks
    print("\n--- INITIATING ML PIPELINE ---")

    if not received_chunks:
        print(" ❌ ML Pipeline Error: No data chunks to process.")
        return
    full_sensor_df = pd.concat(received_chunks, ignore_index=True)
    ir_signal = full_sensor_df['ir'].values
    red_signal = full_sensor_df['red'].values
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    print(f"Data combined. Total samples: {len(ir_signal)}")

    raw_data_filename = os.path.join(OUTPUT_SAVE_DIR, f"raw_data_{timestamp}.csv")
    full_sensor_df.to_csv(raw_data_filename, index=False)
    print(f" ✅ Raw sensor data saved to: {raw_data_filename}")

    plt.figure(figsize=(15, 5))
    plt.plot(ir_signal, color='crimson')
    plt.title(f'IR Signal Waveform - {timestamp}', fontsize=16)
    plt.xlabel('Sample Number')
    plt.ylabel('Raw IR Value')
    plt.grid(True)
    plot_filename = os.path.join(OUTPUT_SAVE_DIR, f"ir_waveform_{timestamp}.png")
    plt.savefig(plot_filename)
    plt.close()
    print(f" ✅ Waveform plot saved to: {plot_filename}")

    temperature = fetch_temperature_from_thingspeak()
    bp_model, stress_model, anomaly_model, state_model, scaler = (
        joblib.load(BP_MODEL_PATH), joblib.load(STRESS_MODEL_PATH),
        joblib.load(ANOMALY_MODEL_PATH), joblib.load(STATE_MODEL_PATH),
        joblib.load(SCALER_PATH)
    )
    try:
        feature_dict = {}
        calculated_spo2 = calculate_spo2(ir_signal, red_signal)
        ppg_signals, info = nk.ppg_process(ir_signal, sampling_rate=SAMPLING_RATE)
        hrv_features = nk.hrv(ppg_signals, sampling_rate=SAMPLING_RATE)
        ppg_analysis = nk.ppg_analyze(ppg_signals, sampling_rate=SAMPLING_RATE)
        feature_dict.update({
            'hrv_DFA_alpha1': hrv_features['HRV_DFA_alpha1'].iloc[0],
            'hrv_MaxNN': hrv_features['HRV_MaxNN'].iloc[0],
            'hrv_Prc20NN': hrv_features['HRV_pNN20'].iloc[0],
            'hrv_CD': hrv_features['HRV_CD'].iloc[0],
```

```
            'hrv_MeanNN': hrv_features['HRV_MeanNN'].iloc[0],
            'hrv_MinNN': hrv_features['HRV_MinNN'].iloc[0],
            'hrv_Prc80NN': hrv_features['HRV_Prc80NN'].iloc[0],
            'ppg_rate_mean': ppg_analysis['PPG_Rate_Mean'].iloc[0],
            'HR_mean': ppg_analysis['PPG_Rate_Mean'].iloc[0],
            'HR_median': ppg_analysis['PPG_Rate_Mean'].iloc[0],
            'HR_std': hrv_features['HRV_SDNN'].iloc[0],
            'SPO2_mean': calculated_spo2,
            'SPO2_median': calculated_spo2,
            'TEMP_mean': temperature,
            'TEMP_median': temperature
        })

        live_features_df = pd.DataFrame([feature_dict])
        live_features_aligned = live_features_df.reindex(columns=TOP_15_FEATURES, fill_value=0)
        live_features_scaled = scaler.transform(live_features_aligned)

        pred_bp = bp_model.predict(live_features_aligned)
        pred_stress = stress_model.predict(live_features_aligned)[0]
        pred_anomaly = anomaly_model.predict(live_features_aligned)[0]
        pred_state = state_model.predict(live_features_aligned)[0]

        results_to_log = {
            'spo2': calculated_spo2, 'hr': feature_dict['HR_mean'],
            'sbp': pred_bp[0][0], 'dbp': pred_bp[0][1],
            'stress': pred_stress, 'anomaly': pred_anomaly, 'state': pred_state
        }
        log_results_to_thingspeak(results_to_log)
    except Exception as e:
        print(f"❌ ML Pipeline execution failed after data saving. Error: {e}")

    received_chunks = []
    print("--- ML PIPELINE COMPLETE. Ready for next cycle. ---")

print("✅ Core functions defined.")

print("\n--- [4/7] Defining Ngrok and web server functions ---")

def setup_ngrok_and_update_thingspeak():
    try:
        ngrok.kill()
        conf.get_default().auth_token = NGROK_AUTH_TOKEN
        port = 5000
        public_url = ngrok.connect(port).public_url
        print(f"🚀 New Ngrok Tunnel URL: {public_url}")
        update_url =
(f"https://api.thingspeak.com/update?api_key={RPI_WRITE_API_KEY}&channel_id={RPI_RESULTS_CHANNEL_ID
}"
                  f"&field7={public_url}")
        requests.get(update_url, timeout=15)
        print("✅ Ngrok URL successfully sent to ThingSpeak (Field 7).")
        return True
    except Exception as e:
        print(f"❌ Critical Error setting up Ngrok or updating ThingSpeak: {e}")
        return False
```

```
@app.route('/data', methods=['POST'])
def receive_data():
    global received_chunks
    try:
        data = request.get_json()
        ir_data, red_data = data.get('ir_data'), data.get('red_data')
        if ir_data is None or red_data is None: return jsonify({"status": "error", "message": "Invalid JSON"}), 400
        chunk_df = pd.DataFrame({'ir': ir_data, 'red': red_data})
        received_chunks.append(chunk_df)
        chunk_count = len(received_chunks)
        print(f"✅ Chunk {chunk_count}/{total_chunks_expected} received!")
        if chunk_count >= total_chunks_expected:
            threading.Thread(target=run_ml_pipeline).start()
        return jsonify({"status": "success", "message": f"Chunk {chunk_count} received"})
    except Exception as e:
        return jsonify({"status": "error", "message": str(e)}), 500

print("✅ Web server and communication functions defined.")

print("\n--- [5/7] Starting the application ---")
print("⚠ Make sure you have updated the NGROK_AUTH_TOKEN and ThingSpeak API keys.")

if setup_ngrok_and_update_thingspeak():
    print("\n🌍 Server is running and waiting for data from ESP32...")
    app.run(host="0.0.0.0", port=5000)
else:
    print("\n❌ Application failed to start due to Ngrok/ThingSpeak setup error.")
```

8.1.3. Machine Learning Model Training Script

```
# Define input and output directories
input_dir = "/home/ubuntu/vitalsdb_preprocessed_aligned_bp_scaled"
output_dir = "/home/ubuntu/vitalsdb_features_bp"
os.makedirs(output_dir, exist_ok=True)

# Define parameters
pleth_sampling_rate = 100 # Hz, as preprocessed
bp_waveform_sampling_rate = 100 # Hz, as preprocessed
window_size_seconds = 30 # Window size for feature extraction
step_size_seconds = 10   # Step size for sliding window

window_size_samples = window_size_seconds * pleth_sampling_rate # Assuming both waveforms are 100Hz
step_size_samples = step_size_seconds * pleth_sampling_rate

expected_hrv_cols = [
    # Time domain
    "MeanNN", "SDNN", "SDANN1", "SDNNI1", "SDANN2", "SDNNI2", "SDANN5", "SDNNI5",
    "RMSSD", "SDSD", "CVNN", "CVSD", "MedianNN", "MadNN", "MCVNN", "IQRNN",
    "Prc20NN", "Prc80NN", "pNN50", "pNN20", "MinNN", "MaxNN", "HTI", "TINN",
    # Freq domain
    "ULF", "VLF", "LF", "HF", "VHF", "LFHF", "LFn", "HFn", "LnHF",
    # Nonlinear
    "SD1", "SD2", "SD1SD2", "S", "CSI", "CVI", "CMSE", "CD", "SampEn", "ShanEn",
    "FuzzyEn", "MSEn", "CMSEn", "RCMSEn", "CDEn", "ApEn", "PALEn", "SCEoEn",
```

```
    "SCLEn", "PFD", "KFD", "LZC", "DFA_alpha1", "DFA_alpha2", "MFDFA_alpha1_Exp",
    "MFDFA_alpha1_Dim", "MFDFA_alpha2_Exp", "MFDFA_alpha2_Dim"
]
# Base features from NeuroKit + Basic ART stats + Basic Numeric Stats (excluding BP targets)
expected_feature_cols_base = [f"hrv_{col}" for col in expected_hrv_cols] + \
                             ["rsp_rate_mean", "ppg_rate_mean"] + \
                             ["ART_mean", "ART_median", "ART_std"] + \
                             ["HR_mean", "HR_median", "HR_std"] + \
                             ["SPO2_mean", "SPO2_median", "SPO2_std"] + \
                             ["TEMP_mean", "TEMP_median", "TEMP_std"]

# Target columns (numeric BP values averaged over the window)
target_bp_cols = ["TARGET_SBP", "TARGET_DBP", "TARGET_MBP"]

# Find all preprocessed case files
case_files = glob.glob(os.path.join(input_dir, "case_*_preprocessed_aligned_bp.csv"))
print(f"Found {len(case_files)} preprocessed case files (with BP) in {input_dir}")
```

8.1.4. Web Frontend (index.html)

```
<!DOCTYPE html>
<html lang="en" class="dark-theme">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PenthuAura Health Monitor</title>
    <!-- Link to the external stylesheet -->
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">

        <!-- Theme Switcher -->
        <div class="theme-switcher" id="theme-switcher" title="Toggle Theme">
            <div class="icon icon-sun">☼</div>
            <div class="icon icon-moon">☽</div>
        </div>

        <header>
            <h1>PenthuAura</h1>
            <p class="subtitle">Your Personal Health Guardian, Inspired by Ancient Wisdom</p>
        </header>

        <nav class="nav-tabs">
            <button class="tab-btn active" data-tab="dashboard">Dashboard</button>
            <button class="tab-btn" data-tab="config">Configuration</button>
            <button class="tab-btn" data-tab="history">History</button>
            <button class="tab-btn" data-tab="about">About</button>
        </nav>

        <!-- Dashboard Tab -->
        <div id="dashboard" class="tab-content active">
            <div class="status-grid">
                <div class="status-card vitals">
                    <h3>Current Vitals</h3>
```

```
            <div class="vital-item"><span class="vital-label">SpO2</span><span id="spo2-value" class="vital-value">--</span></div>
            <div class="vital-item"><span class="vital-label">Heart Rate</span><span id="hr-value" class="vital-value">--</span></div>
            <div class="vital-item"><span class="vital-label">Temperature</span><span id="temp-value" class="vital-value">--</span></div>
            <div class="vital-item"><span class="vital-label">Blood Pressure</span><span id="bp-value" class="vital-value">--</span></div>
          </div>

          <div class="status-card system">
            <h3>Device & Health Status</h3>
            <div class="status-item"><span class="status-label">WiFi</span><span id="wifi-status" class="status-value">--</span></div>
            <div class="status-item"><span class="status-label">Health State</span><span id="health-state-value" class="status-value">--</span></div>
            <div class="status-item"><span class="status-label">Stress Level</span><span id="stress-value" class="status-value">--</span></div>
            <div class="status-item"><span class="status-label">Anomaly Status</span><span id="anomaly-status-value" class="status-value">--</span></div>
          </div>
        </div>
      </div>

      <!-- Configuration Tab -->
      <div id="config" class="tab-content">
        <div class="config-section">
          <h3>WiFi Configuration</h3>
          <div class="form-group">
            <label for="wifi-ssid">WiFi SSID:</label>
            <input type="text" id="wifi-ssid" placeholder="Enter WiFi network name">
          </div>
          <div class="form-group">
            <label for="wifi-password">WiFi Password:</label>
            <div class="password-wrapper">
              <input type="password" id="wifi-password" placeholder="Enter WiFi password">
              <span class="password-toggle" id="password-toggle">👁</span>
            </div>
          </div>
          <button id="scan-wifi-btn" class="btn secondary">Scan Networks</button>
          <div id="wifi-networks" class="wifi-list"></div>
        </div>
        <div class="config-section">
          <h3>Cycle Configuration</h3>
          <div class="form-group">
            <label for="cycle-interval">Cycle Interval (minutes):</label>
            <input type="number" id="cycle-interval" min="1" max="1440" value="10">
          </div>
        </div>
        <button id="save-config-btn" class="btn primary">Save Configuration</button>
        <button id="reboot-btn" class="btn secondary">Exit Setup & Reboot</button>
        <div id="config-status" class="status-message"></div>
      </div>

      <!-- History Tab -->
      <div id="history" class="tab-content">
```

```
        <div class="history-section">
          <h3>Measurement History</h3>
          <div id="history-container"><p>Loading history...</p></div>
          <p class="history-scroll-hint">Scroll right to see more data.</p>
        </div>
      </div>

      <!-- About Tab -->
      <div id="about" class="tab-content">
        <div class="about-section">
          <h3>About PenthuAura</h3>
          <h4>The Vision</h4>
          <p>PenthuAura is a bridge between ancient wisdom and modern technology. Inspired by Penthu, the revered chief physician of Pharoah Akhenaten, this device embodies a dedication to healing and well-being. The name "PenthuAura" represents the protective and caring influence (the "aura") this device provides, ensuring the wearer is always under the watchful eye of a modern-day Penthu.</p>

          <h4>Project Workflow</h4>
          <div class="workflow-step">
            <strong>Phase 1: Data Collection (The ESP32 Device)</strong>
            <p>The wearable device continuously gathers raw sensor data, including PPG waveforms (for SpO2 and HR) and temperature readings. It then securely transmits this data in chunks to our central processing server.</p>
          </div>
          <div class="workflow-step">
            <strong>Phase 2: Analysis & Prediction (The Raspberry Pi Server)</strong>
            <p>Our powerful server, powered by a Raspberry Pi, receives the raw data. It applies advanced signal processing and runs our custom-trained AI models to calculate vitals and predict health metrics like Blood Pressure, Stress Levels, and the overall Health State (Normal, Moderate, or Critical).</p>
          </div>
          <div class="workflow-step">
            <strong>Phase 3: Visualization & Cloud (ThingSpeak & Web UI)</strong>
            <p>The RPi uploads all final results to the ThingSpeak cloud. The ESP32 device then fetches this processed data, displaying it for the user on the built-in LCD screen and this interactive web dashboard.</p>
          </div>

          <h4>Our Team</h4>
          <p>This device is the culmination of a final graduation project by a dedicated team from the Canadian International College (CIC), School of Engineering, Department of Communication and Electronics. As the cohort of 2020, we proudly present this project in 2025.</p>
          <ul class="team-list">
            <li>Diaa Ahmed</li>
            <li>Abdulrahman Nageh</li>
            <li>Mohanad Tarek</li>
            <li>Yehia Mohammed</li>
          </ul>
          <p class="supervisor">Under the supervision of Dr. Omar Sabry</p>

          <p class="footer-text">PenthuAura © 2025</p>
        </div>
      </div>
    </div>
```

8.2. Appendix B: Hardware Datasheets

This appendix contains the official datasheets for the primary electronic components used in the project.

- B.1: ESP32-WROOM-32 Datasheet.
- B.2: MAX30102 Datasheet.
- B.3: MAX30205 Datasheet.

8.3. Appendix C: Bill of Materials (BOM)

The table below lists all electronic components used for the construction of one PenthuAura prototype.

Component	Quantity	Description
ESP32-WROOM-32 Dev Kit	1	Microcontroller with Wi-Fi and Bluetooth
MAX30102 Module	1	PPG and SpO2 Sensor Breakout Board
MAX30205 Module	1	High-Accuracy Temperature Sensor Breakout Board
16x2 Character LCD	1	Standard HD44780 compatible display
10kΩ Potentiometer	1	For LCD contrast adjustment
Common Anode RGB LED	1	5mm, 4-pin
Active Buzzer	1	5V operating voltage
Tactile Push Button	1	Momentary switch
Resistors (220Ω)	3	Current limiting for RGB LED channels
Dot-Matrix PCB (Perfboard)	1	For prototype assembly
Header Pins & Wires	various	For connections

8.4. Appendix D: Raspberry Pi systemd Service File

This file (penthuaura.service) is placed in /etc/systemd/system/ on the Raspberry Pi to ensure the backend server runs as a persistent service.

```
[Unit]
Description=PenthuAura Health Monitoring Server
After=network-online.target

[Service]
User=penthuaura
Group=penthuaura
WorkingDirectory=/home/penthuaura/Models
ExecStart=/home/penthuaura/start_penthuaura.sh
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```