19L820 – PROJECT WORK II

CUFFLESS BLOOD PRESSURE MONITORING SYSTEM

KAMAL S (Roll No: 20L215)

SHARAN S (Roll No: 20L237)

ASHWIN B (Roll No: 21L431)

Report submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: ELECTRONICS AND COMMUNICATION ENGINEERING

of Anna University

APRIL 2024

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

CUFFLESS BLOOD PRESSURE MONITORING SYSTEM

Bonafide record of work done by

KAMAL S (Roll No: 20L215)

SHARAN S (Roll No: 20L237)

ASHWIN B (Roll No: 21L431)

Report submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: ELECTRONICS AND COMMUNICATION ENGINEERING

of Anna University

APRIL 2024

..……	……..…….
DR.K. VASANTHAMANI	**DR.V. KRISHNAVENI**
Faculty guide	**Head of the Department**

Certified that the candidate was examined in the viva-voce examination held on ……

……..	……..
(Internal Examiner)	**(External Examiner)**

CONTENTS

ACKNOWLEDGEMENT

We extend our sincere thanks to **Dr. K. PRAKASAN**, Principal, PSG College of Technology, for his kind patronage.

We are indebted to **Dr. V. KRISHNAVENI**, Professor and Head, Department of Electronics and Communication Engineering, for her continued support and motivation.

We express our deepest gratitude to our project guide, **Dr. K. VASANTHAMANI**, Assistant Professor (SI. Gr.), Department of Electronics and Communication Engineering, for her constant motivation, direction and guidance throughout the course of our project work.

Our sincere thanks to Programme Coordinator, **Dr. M. SANTHANALAKSHMI**, Associate Professor, Department of Electronics and Communication Engineering, Committee Members **Dr. P. VISALAKSHI,** Professor, **Dr**. **T. KESAVAMURTHY,** Professor and **Dr. G. SRIVATSUN**, Associate Professor whose valuable guidance and continuous encouragement throughout the course made it possible to complete this dissertation work successfully.

Our deepfelt thanks to the technical faculty who helped by providing lab facilities. Finally, we thank all our student colleagues who have helped us in completing this project without any hassles.

SYNOPSIS

Hypertension, a matter of global health significance, impacts more than 1.3 billion people across the globe, and it is projected to increase to 1.5 billion by the year 2025. The conventional approaches for measuring blood pressure (BP), such as auscultatory and oscillometric techniques, possess certain limitations such as interrupted measurement cycles, potential discomfort for patients, lack of remote monitoring and the inability to provide continuous data. To overcome these drawbacks, this work introduces a cuffless BP monitoring system that integrates hardware and software components for continuous, comfortable, and non-invasive BP tracking.

The system uses an AD8232 ECG sensor to capture Electrocardiogram (ECG) signals and a pulse oximeter sensor to gather Photoplethysmogram (PPG) signals. These physiological signals are then processed wirelessly on a secure cloud server via Wi-Fi. To predict the blood pressure (BP) values from the synchronized ECG and PPG data streams DNN, RNN, LSTM and NARX architectures were evaluated on the server-side. The Long Short-Term Memory (LSTM) model with two LSTM layers and one Dense layer emerged as the most efficient model after field testing. Arterial BP estimation algorithms were proposed to estimate Systolic and Diastolic BP. The server stores the calculated blood pressure (BP) measurements, enabling uninterrupted monitoring without the need for cuffs or interrupting the user's activities. A web application is designed using Flask and React frameworks to facilitate authorized remote access and visualization of historical BP data. The user information is securely stored in a database with a unique ID.

The BP estimation model exhibited a Mean Absolute Error (MAE) of 7.02 mmHg for Arterial BP when validated on MIMIC-III Dataset. This system was deployed and tested on 30 subjects achieving an MAE of 4.06 mmHg for Systolic BP and 4.32 mmHg for Diastolic BP, with their BP readings compared to those obtained from a digital sphygmomanometer. The computed MAE for Systolic and Diastolic BP falls under the maximum standard allowable error mentioned by Association for Advancement of Medical Instrumentation i.e. MAE < 5 mmHg .The proposed cuffless BP monitoring system offers notable advantages over traditional techniques, including enhanced user comfort, continuous data acquisition, and improved accessibility to BP monitoring for better preventive healthcare and management of chronic conditions.

TABLE OF FIGURES

TABLE OF TABLES

CHAPTER 1

INTRODUCTION

In the realm of healthcare, the absence of a practical, accurate, and non-invasive method for continuous blood pressure (BP) monitoring represents a significant challenge. The conventional cuff-based techniques, while informative, are marred by discomfort and impracticality for regular monitoring. This limitation results in sporadic measurements and restricted data accessibility, consequently posing serious challenges for individuals managing hypertension or other cardiovascular conditions. The inability to consistently monitor BP levels can hinder effective condition management, making it imperative to seek innovative solutions in this regard. Cuffless blood pressure monitoring is needed to bridge this crucial gap in healthcare, offering a convenient, non-invasive, and continuous alternative that can enhance the management of BP-related conditions and provide individuals with a more comprehensive and user-friendly method for tracking their cardiovascular health.

1.1 THE ESSENTIALS OF THE HEART AND BLOOD PRESSURE

1.1.1 Structure of Heart

The human heart is a complex organ, serving as the central pump for the circulatory system. It comprises four chambers, each with a specific role. These chambers are divided into two atria.

Atria: The atria are the upper chambers of the heart. The left atrium receives oxygenated blood from the lungs through the pulmonary veins. Simultaneously, the right atrium receives deoxygenated blood returning from the body through the superior and inferior vena cava. These atria serve as collection chambers, accepting blood from their respective sources.

Ventricles: The ventricles, situated below the atria, are the powerful pumping chambers. The left ventricle is responsible for pumping oxygen-rich blood into the aorta, which carries it to the entire body, providing vital oxygen and nutrients to the body's tissues. The right ventricle pumps deoxygenated blood into the pulmonary artery, leading to the lungs, where it becomes oxygenated again.

Valves: Within the heart, there are specialized valves that play a critical role in maintaining the orderly flow of blood through its chambers. Two types of valves are particularly important:

- Atrioventricular (AV) Valves: These are the tricuspid valve on the right side and the bicuspid (or mitral) valve on the left side. The tricuspid valve separates the right

atrium from the right ventricle, and the bicuspid valve separates the left atrium from the left ventricle.

- Semilunar Valves: The semilunar valves are the pulmonary valve (on the right side) and the aortic valve (on the left side these valves ensure that blood is pumped out of the heart and into the respective circulatory pathways without regurgitation.

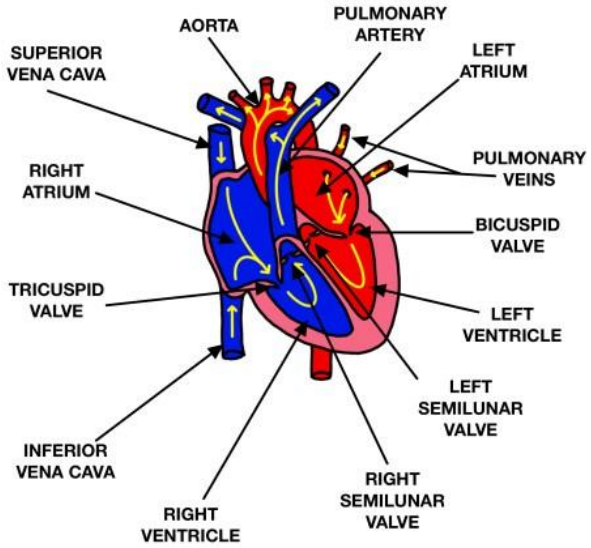


Figure 1.1: Structure of Heart

1.1.2 The Cardiac Cycle

The cardiac cycle is a rhythmic sequence of events that occurs with every heartbeat, and it plays a central role in maintaining blood circulation throughout the body. This cycle is divided into two main phases: systole and diastole, each serving a distinct purpose.

Systole: Systole is the phase of the cardiac cycle characterized by the contraction of the heart's muscular chambers, specifically the ventricles. During this phase, the ventricles contract forcefully, generating the pressure needed to pump blood into the pulmonary artery (right ventricle) and aorta (left ventricle). This phase is also referred to as ventricular systole and is essential for propelling oxygenated blood to the body's tissues and deoxygenated blood to the lungs. Systole typically starts when the atria contract, pushing the remaining blood into the ventricles, and culminates when the aortic and pulmonary valves open to eject blood into the respective arteries.

Diastole: Diastole is the phase of the cardiac cycle marked by relaxation. During diastole, the heart muscles, particularly the ventricles, relax to prepare for the next cycle of contraction. In this phase, the heart chambers passively fill with blood. Diastole is further divided into two stages: early diastole, when blood flows into the ventricles and the atrioventricular valves open, and late diastole, during which the atria contract to push the last portion of blood into the ventricles before the next systole. Diastole is critical as it allows the heart to refill with blood and ensures an adequate supply of oxygen and nutrients to the cardiac muscle itself.

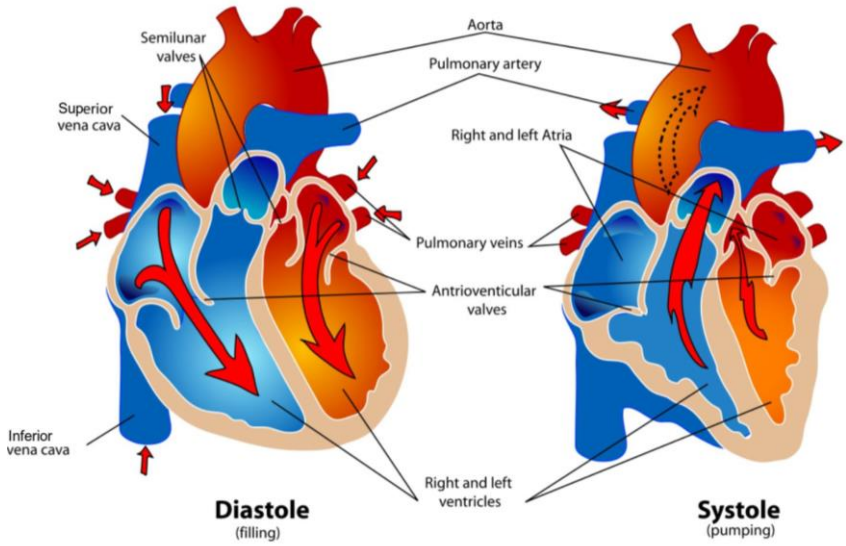


Figure 1.2: The Cardiac Cycle

Blood Flow

The sequence of blood flow through the heart is essential for the efficient functioning of the circulatory system. Here's a detailed description of this process:

- Right Atrium: Deoxygenated blood from the body returns to the right atrium via two large veins, the superior and inferior vena cava. This blood is low in oxygen and rich in carbon dioxide, a waste product from cellular metabolism.
- Right Ventricle: During diastole, the right atrium contracts, pushing the deoxygenated blood into the right ventricle through the tricuspid valve. The right ventricle stores this blood until it's ready to be sent to the lungs.
- Lungs for Oxygenation: When the right ventricle contracts during systole, the pulmonary valve opens, allowing the deoxygenated blood to flow into the pulmonary artery. This artery carries the blood to the lungs where it can exchange carbon dioxide for oxygen. This oxygen-rich blood returns to the heart.
- Left Atrium: Oxygenated blood returns to the heart and enters the left atrium via the pulmonary veins. The left atrium contracts, pushing the blood into the left ventricle through the bicuspid valve (mitral valve).
- Left Ventricle: During systole, the left ventricle contracts forcefully, and the aortic valve opens, allowing oxygen-rich blood to flow into the aorta. The aorta is the body's main artery, distributing oxygenated blood to all organs and tissues.

This synchronized process ensures that oxygen-rich blood is pumped to nourish the cells of the body, while deoxygenated blood is sent to the lungs for reoxygenation. This

continuous cycle of systole and diastole maintains the life-sustaining circulation of blood throughout the entire body.

Understanding the cardiac cycle and the sequence of blood flow through the heart is fundamental to appreciating how the heart efficiently delivers oxygen and nutrients to tissues while managing the exchange of gases, such as oxygen and carbon dioxide, in the lungs. Any disruption in this cycle can lead to cardiac issues and impact overall health.

1.1.3 Blood Pressure

Blood pressure is a fundamental physiological parameter that represents the force exerted by the circulating blood against the walls of the arteries. This force is generated by the pumping action of the heart and is necessary to propel blood throughout the entire circulatory system. Blood pressure is a critical aspect of cardiovascular health and is measured in millimetres of mercury (mmHg). Typically, blood pressure is expressed as a ratio of two values, the systolic pressure over diastolic pressure (e.g., 120/80 mmHg). This numerical representation is used to gauge the pressure levels in the arteries at different phases of the cardiac cycle.

In [1] Arterial Blood Pressure is the most commonly measured type of blood pressure. It reflects the force exerted by the left ventricle of the heart during two distinct phases of the cardiac cycle:

In [1] Systolic Pressure represents the higher number in a blood pressure reading (range: 90 to 120 mmHg). Systolic pressure occurs during the systole phase of the cardiac cycle when the left ventricle contracts vigorously. It generates a surge of pressure that forces blood out of the heart and into the arteries, creating a momentary peak in arterial pressure. This peak pressure is essential for pushing oxygenated blood into the systemic circulation, supplying the body's organs and tissues.

In [1] Diastolic Pressure represents the lower number in a blood pressure reading (range: 60 to 80 mmHg). It occurs during diastole, the relaxation phase of the cardiac cycle when the heart is at rest between beats. During diastole, the left ventricle refills with blood. Diastolic pressure reflects the residual pressure within the arteries when the heart is not actively contracting. It ensures a continuous, steady flow of blood to the body's tissues even when the heart is at rest.

1.1.4 Blood Vessels

Arteries are blood vessels responsible for carrying oxygenated blood away from the heart and distributing it to the body's various tissues and organs. Arteries have a thick and elastic wall, allowing them to expand and contract in response to changes in blood pressure, thereby helping to maintain a steady flow of blood throughout the circulatory system.

Veins are the counterpart to arteries, as they transport deoxygenated blood from the body's tissues back to the heart. Unlike arteries, veins typically have thinner walls and are less muscular. They are able to operate with less force because they carry deoxygenated blood at

lower pressure levels. These valves ensure that blood flows in only one direction, preventing any backflow. This mechanism is particularly crucial in the legs, where it helps blood return to the heart against the force of gravity.

1.1.5 Blood Pressure Measurement

Sphygmomanometer

In [2] A sphygmomanometer is a classic and widely recognized device used for measuring blood pressure. It consists of three primary components:

Inflatable Cuff: The cuff is wrapped around the upper arm and is inflated to momentarily stop the flow of blood in the brachial artery. This interruption is essential for the accurate measurement of blood pressure.

Pressure Gauge: The pressure gauge, often in the form of a column of mercury or a digital display, records the pressure within the cuff. It provides the numerical values used to determine systolic and diastolic blood pressure.

Stethoscope: A stethoscope is used to listen to the sounds of blood flow in the artery during the measurement process. These sounds, known as Korotkoff sounds, help healthcare providers identify the point at which the blood starts flowing (systolic pressure) and when it ceases (diastolic pressure). This manual approach is known as auscultatory blood pressure measurement.

Figure 1.3: Sphygmomanometer

Automated Blood Pressure Monitors

In [3] Modern digital devices have revolutionized the way blood pressure is measured, offering convenience and ease of use for routine monitoring. These automated blood pressure monitors employ oscillometric methods, which involve measuring pressure changes in the cuff as the blood pulses through the artery. Key features of these devices include:

Inflatable Cuff: Similar to the traditional sphygmomanometer, automated blood pressure monitors include an inflatable cuff that is typically wrapped around the upper arm or wrist.

Automated Inflation and Deflation: Unlike manual methods, these devices automatically inflate and deflate the cuff, eliminating the need for manual pressure adjustments. This makes them user-friendly and suitable for self-monitoring.

Digital Display: The blood pressure readings are displayed on a digital screen, providing systolic and diastolic values along with pulse rate information. Many devices also store previous measurements for tracking trends over time.

Figure 1.4: Automated Blood Pressure Monitors

1.2 CUFFLESS BLOOD PRESSURE MONITORING

Cuffless blood pressure monitoring is an innovative and non-invasive approach that is gaining prominence in healthcare. Unlike traditional cuff-based methods, which involve intermittent pressure measurements, cuffless methods aim to provide continuous data without the discomfort and disruption caused by cuffs. This means that patients can have their blood pressure continuously monitored without the need for periodic cuff inflations and deflations. Continuous monitoring can offer a more comprehensive picture of blood pressure trends and variations over time, which is valuable in the management of cardiovascular health.

1.2.1 Principles of Cuffless Blood Pressure Monitoring

In [4] Photoplethysmography (PPG) is a fundamental technology behind cuffless blood pressure monitoring. PPG involves measuring changes in blood volume within tissues, typically through the skin. By the pulsatile nature of arteries specialized PPG sensors use light to detect variations in blood volume. This technology is often utilized in wearable devices like smartwatches and patch-based monitors for continuous monitoring of blood volume.

In [5] Ballistocardiography (BCG) is another technique used in cuffless monitoring. BCG sensors measure the recoil force of the body due to the ejection of blood from the heart. When the heart contracts and ejects blood into the arteries, it generates a force that can be detected by BCG sensors. These sensors can derive blood pressure data from the force

measurements. BCG technology doesn't rely on direct contact with the skin, making it suitable for non-contact monitoring.

1.2.3 Accuracy and Validation

The accuracy of cuffless blood pressure monitoring devices is a critical consideration for their clinical utility and hence clinical studies are essential to establish their reliability, Validation of cuffless devices against traditional cuff-based measurements is essential to ensure that the cuffless methods provide accurate and consistent results. Clinical validation helps build trust in these innovative monitoring technologies.

In [6] Cuffless devices often require calibration to account for individual physiological variations. To Calibrate adjust the device's measurements to match a known standard. Calibration methods can be essential to ensure that the measurements obtained from cuffless devices are accurate and reliable. The choice of calibration methods and their impact on measurement accuracy is a significant consideration in the development and use of cuffless blood pressure monitoring devices.

1.2.4 Advantages

Continuous Monitoring

In [6] One of the primary advantages of cuffless blood pressure monitoring is its capability for continuous monitoring. Unlike traditional cuff-based methods, which provide periodic snapshots of blood pressure, cuffless methods offer the ability to capture variations and trends in blood pressure over extended periods. This continuous monitoring can be particularly beneficial for understanding how blood pressure responds to daily activities and changes in health.

Reduced Discomfort

The absence of cuffs in cuffless monitoring reduces discomfort for patients. Cuff-based measurements can sometimes be uncomfortable and may even induce anxiety, leading to elevated blood pressure readings (white coat syndrome). Cuffless methods are less invasive, making patients more compliant and willing to engage in long-term monitoring. The comfort they offer can improve the patient experience and encourage regular monitoring.

Remote Monitoring

In [6] Cuffless devices can transmit data wirelessly, allowing for remote monitoring by healthcare providers. This feature is particularly valuable for telehealth applications, where patients can have their blood pressure monitored and shared with healthcare professionals from the comfort of their homes. Remote monitoring enhances patient care by enabling healthcare providers to assess patients' blood pressure data and make informed decisions regarding treatment and medication adjustments.

1.2.5 Applications of Cuffless Blood Pressure Monitoring

- Improving hypertension awareness and management

Cuffless BP monitoring can help improve hypertension awareness, treatment, and management, which can lead to early prediction of cardiovascular events.

- Improving antihypertensive drug treatment compliance

Cuffless BP monitoring can help improve antihypertensive drug treatment compliance and hypertension control rates by continually revealing high BP in individual patients.

- Increasing self-awareness of hypertensive patients

The easy measurement method and the device portability (integrated in a smartphone) may increase the self-awareness of hypertensive patients.

- Cardiovascular Health Tracking

Continuous blood pressure monitoring contributes to a better understanding of overall cardiovascular health. It allows for early detection of abnormalities and can be a valuable tool in the prevention and management of heart-related conditions.

- Remote Patient Monitoring

Cuffless devices have immense potential in remote patient monitoring. They enhance patient care in both clinical and home settings by allowing healthcare providers to remotely track and manage patients' blood pressure. This is especially valuable in scenarios where regular clinic visits may be challenging, such as during a pandemic.

Cuffless blood pressure monitoring represents a significant advancement in the field of healthcare, offering non-invasive, continuous monitoring with advantages including comfort, convenience, and the potential for remote patient care.

1.3 ORGANIZATION OF THE REPORT:

- The introduction to the essentials of the Heart and Blood Pressure, Cuffless Blood Pressure Monitoring are discussed in Chapter 1.
- Chapter 2 includes the literature survey which describes the key concepts and special features present in each of the research papers.
- The conventional and existing systems are given in Chapter 3.
- The proposed methodology along with implementation results are elaborately discussed in Chapter 4.
- Conclusions and future works are provided clearly in Chapter 5.

CHAPTER 2

LITERATURE SURVEY

In [6], **"*Non-invasive, wearable multi biosensors for continuous, long-term monitoring of blood pressure via internet of things applications*"**, This paper proposes a method of multimodal wearable biosensor device designed for continuous blood pressure (BP) monitoring during physical exercise and their integration into Internet of Things (IoT) applications. The focus of this survey is the exploration of a BP monitoring system that boasts the capability to seamlessly connect to a cloud server. Further investigate the effectiveness of this novel monitoring device by reviewing studies that have validated its performance in capturing the natural variability of BP during various cycling tests. This evaluation underscores the system's viability for continuous, long-term BP monitoring, cementing its significance in the field of health monitoring and IoT applications. While the paper highlights the potential of the BP monitoring system in IoT applications, one limitation is the need for further validation in diverse demographic groups and real-world settings to ensure its effectiveness across different populations and environments.

In [7], ***"Real time system on chip based wearable cardiac activity monitoring sensor",*** This paper explores the method to advance the field of wireless IoT health monitoring by designing an innovative system tailored for remote cuff-less blood pressure (BP) monitoring. The results unveiled compelling insights into the potential of edge computing in this domain. Notably, it was found that implementing edge computing technology led to a 15% reduction in application latency, an impressive 85% decrease in the total transmitted data volume, and a notable 19% reduction in power usage when employing a batch size of 64 ECG samples. These findings underscore the pivotal role of edge computing in mitigating various operational challenges, thereby enhancing system efficiency. Moreover, this study ventures into the utilization of edge computing as a strategic approach to alleviate processing burdens, extend battery life, and facilitate decentralized data collection. Despite the promising results regarding the benefits of edge computing in reducing latency, data transmission volume, and power usage, the study lacks a detailed analysis of potential security concerns and data privacy issues associated with decentralized data collection and processing at the edge.

In [8], ***"A wireless low-power single-unit wearable system for continuous early warning score calculation",*** This paper proposes a method on a groundbreaking device designed for the comprehensive monitoring of vital parameters, representing a significant advancement in healthcare technology. This innovative device captures a wide array of critical data, including ECG (Electrocardiogram), PPG (Photoplethysmogram), bioimpedance, body

temperature, and acceleration. The captured data is seamlessly transmitted via WIFI connectivity to a central server. At the server end, sophisticated algorithms come into play to calculate vital parameters, gauge the individual's activity level, and assess their posture, rendering a comprehensive profile of the patient's health status. Although the comprehensive monitoring device discussed in the paper offers a wide range of vital parameter measurements, a limitation here is the absence of extensive clinical validation studies to confirm the accuracy and reliability of the device's measurements compared to gold-standard clinical instruments.

In [9], ***"Wearable blood pressure monitoring based on bio-impedance and photoplethysmography sensors on the arm"***, This paper explores the methodology innovative wearable system designed for real-time beat-to-beat blood pressure (BP) monitoring, introducing a novel approach that utilizes pulse transit time (PTT). The PTT metric is derived by calculating the time difference between the pulse signal recorded by the bio-impedance (BI) sensor and the photoplethysmography (PPG) sensor. The results of this study offer compelling evidence of the efficacy of the proposed method. Specifically, the beat-to-beat pulse wave velocity (PWV) derived from this innovative approach exhibits a strong correlation with the measured beat-to-beat reference systolic and diastolic BP values. This correlation underscores the potential of the PTT-based monitoring system as a promising avenue for accurate and real-time blood pressure assessment This approach presents new possibilities for non-invasive, real-time assessment, thereby contributing significantly to the field of cardiovascular health and remote patient monitoring. While the PTT-based monitoring system shows promise for real-time blood pressure assessment, a limitation is the need for further investigation into its performance in scenarios involving dynamic changes in blood pressure, such as during exercise or stress, to ensure its accuracy and reliability in various physiological conditions.

In [10], ***"Localized bioimpedance measurements with the MAX3000x integrated circuit: Characterization and demonstration"***, This paper discusses the innovative methodology that introduces a hybrid approach (Photoplethysmogram) signals with a recurrent neural network (RNN) to address the challenge of blood pressure estimation. Notably, the RNN model employs a bidirectional LSTM (Long Short-Term Memory) layer, a strategic choice aimed at mitigating the impact of past and future physiological changes on the estimation process. This combination harnesses the power of deep learning, incorporating LSTM and ReLU layers, to enhance model performance and reduce errors in the estimation process. The study's results underscore the potential of this hybrid model, demonstrating its ability to estimate blood pressure with remarkable accuracy. This survey sheds light on the significance of this innovative approach, emphasizing its role in advancing non-invasive blood pressure estimation methods and its potential applications in the field of healthcare and medical diagnostics. The limitation in this paper is the generalizability of the hybrid model

across different demographic groups and physiological conditions, necessitating further validation in diverse populations and real-world settings to assess its robustness and reliability.

In [11], ***"Neckband-based continuous blood pressure monitoring device with offset-tolerant ROIC",*** The approach of this paper is to enhance the accuracy of cuff-less blood pressure estimation, particularly through the utilization of Pulse Transit Time (PTT). Recognizing the inherent limitations in PTT's accuracy as a standalone metric for blood pressure (BP) estimation, a recent study introduces an innovative model that incorporates heart rate (HR) data into the equation. This model, employing linear regression techniques, is rigorously evaluated using the MIMIC database as the testing ground. The study's findings reveal significant advancements in accuracy compared to conventional PTT-only methods. It underscores the substantial improvements in accuracy and reliability brought about by this novel model, thereby contributing to the evolving landscape of healthcare technology and patient monitoring. While the model incorporating heart rate data alongside PTT demonstrates improved accuracy in blood pressure estimation, a limitation might be the computational complexity of the model, which could hinder its real-time implementation in resource-constrained environments such as wearable devices. Simplification or optimization of the model architecture may be necessary to enable practical deployment in such settings.

In [12], ***"Cuffless blood pressure monitoring from an array of wrist bio-impedance sensors using subject-specific regression models: Proof of concept",*** This Paper discuss the method for continuous and beat-to-beat monitoring of blood pressure (BP), offering a notable departure from traditional office-based BP measurement. The advantages of continuous BP monitoring in predicting future cardiovascular disease are substantial, yet traditional methods reliant on bulky and obtrusive cuffs are impractical for this purpose. The application of pulse transit time (PTT), a prominent cuffless technique for continuous BP monitoring. PTT measures the time taken for a pressure pulse to travel between two points within an arterial vessel and has a significant correlation with BP were explored. The core contribution of this paper is the introduction of an innovative cuffless BP monitoring method utilizing an array of wrist-worn bio-impedance sensors strategically placed on the radial and ulnar arteries. However, in this paper the limitation is the necessity for validation in dynamic conditions such as during physical activity or with individuals with different physiological characteristics to assess the accuracy and reliability of the cuffless BP monitoring method utilizing wrist-worn bio-impedance sensors.

In [13], **"*Blood pressure estimation using a single channel Bio-Impedance Ring Sensor",*** This paper discusses about smart rings, a category of wearable devices offering distinct advantages for continuous physiological monitoring. These rings present an enticing alternative for health monitoring, characterized by their comfort, minimal encumbrance compared to traditional smart wearables, suitability for nocturnal settings, and the ability to maintain ideal sensor-skin contact throughout their use. Of particular significance is their

application in the continuous measurement of blood pressure (BP), a parameter of paramount importance in cardiovascular health management. This survey unveils the integration of a human finger finite element model with extensive experimental data to derive optimal design parameters for electrode placement and sizes, ensuring the highest sensitivity to arterial volumetric changes. This survey underscores the substantial potential of bioimpedance-based smart rings for the precise and continuous estimation of BP, marking a pivotal step towards redefining cardiovascular health monitoring. The challenge in this paper is ensuring consistent and reliable sensor-skin contact over time with smart rings for continuous physiological monitoring, particularly during daily activities or movements, which is crucial for maintaining accurate and continuous monitoring.

In [14], ***"An Affordable Cuff-Less Blood Pressure Estimation Solution",*** This paper introduces a cuff-less hypertension pre-screening device for continuous monitoring of Blood Pressure (BP) and Heart Rate (HR) using Electrocardiogram (ECG) and Photoplethysmogram (PPG) signals. The device, compatible with various platforms like PC, smartphone, and Raspberry Pi, employs kernel regression for real-time BP and HR extraction. Validation against a standard BP monitor confirms its accuracy within acceptable limits set by the Association for the Advancement of Medical Instrumentation. Developed with cost-effective components, the device offers a portable, affordable solution for home and clinic-based health monitoring, with potential for future wearable integration and cloud-based data storage. The limitation of this paper is the need for further validation in clinical settings to ensure the accuracy and reliability of the cuff-less hypertension pre-screening device across a wide range of individuals and health conditions, despite its cost-effective and portable nature.

In [15], "***Cuffless blood pressure meter with mobile application for home-care Service***". This paper addresses the limitations of conventional cuff-based blood pressure monitors by introducing a rechargeable cuffless blood pressure meter based on Pulse Transit Time (PTT) technique using Arduino technology and ECG-PPG correlation. By acquiring signals from Electrocardiogram (ECG) and reflective optical sensors, the device estimates blood pressure through signal processing algorithms. A mobile application facilitates real-time monitoring and trend analysis. This innovation offers a promising alternative for painless continuous blood pressure monitoring, contributing to improved healthcare outcomes. Despite the potential of the rechargeable cuffless blood pressure meter based on PTT technique using Arduino technology for continuous monitoring, a limitation could be the challenge of ensuring user compliance and adherence to proper device placement and usage instructions. Variability in user behaviour may impact the accuracy and reliability of blood pressure measurements, warranting further investigation into user interaction and training protocols to optimize device performance.

INFERENCE:

Advancements in wearable biosensors and IoT integration facilitate seamless blood pressure monitoring, enabling remote healthcare management. While edge computing enhances system efficiency by reducing latency and data transmission volumes, security concerns remain. Comprehensive vital parameter monitoring devices offer extensive health data analysis capabilities, but further clinical validation is necessary. Novel methodologies, such as PTT-based monitoring and hybrid models, hold promise for real-time blood pressure assessment but require validation across diverse populations. Smart wearable devices offer cost-effective solutions for continuous physiological monitoring, yet user compliance and device usage pose ongoing challenges.

CHAPTER 3

CUFFLESS BLOOD PRESSURE MONITORING SYSTEM

This section provides an overview of the systematic approach employed in the development and validation of a cuffless blood pressure (BP) monitoring system. The study utilizes specialized sensors, namely AD8232 for Electrocardiogram (ECG) signals and SEN 11574 for Photoplethysmogram (PPG) signals. The main focus of the research is to leverage deep learning architectures, such as Long Short-Term Memory (LSTM), Recurrent Neural Network (RNN), and Nonlinear Auto-regressive with exogenous inputs (NARX) models, for predicting arterial BP values. Once the appropriate models are selected and algorithms are developed, validation is carried out using the MIMIC-III Dataset. Furthermore, the system is deployed and tested on a group of 30 subjects to assess its real-world performance in comparison to a digital sphygmomanometer.

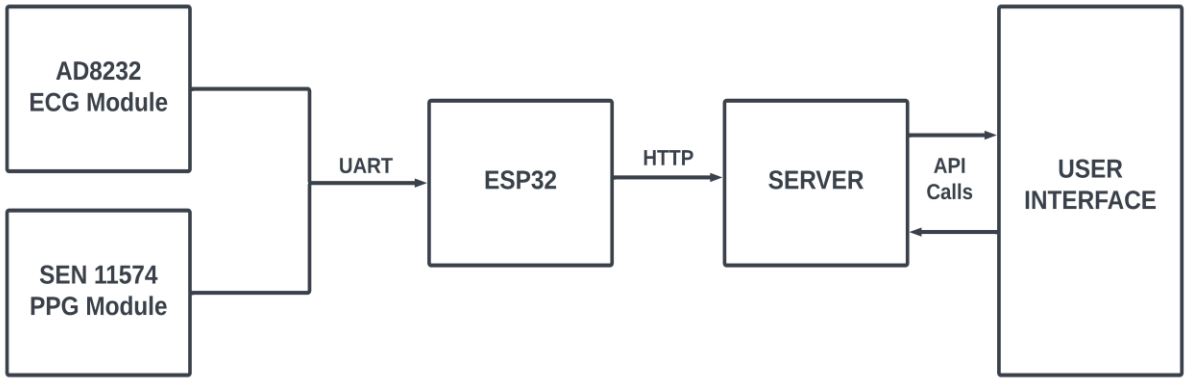


Figure 3.1: ECG, PPG Signal Acquisition & Processing

The block diagram presented in Figure 3.1 depicts the system architecture consisting of several interconnected components. The primary elements include an electrocardiogram (ECG) device, a photoplethysmogram (PPG) device, an ESP32 microcontroller, a server, and a user interface.

The ECG and PPG devices will acquire physiological signals from the user. The ECG device will measure the electrical activity of the heart, while the PPG device will measure the changes in blood volume through optical methods. Both devices will be connected to the ESP32 microcontroller via UART (Universal Asynchronous Receiver-Transmitter) interfaces, facilitating the transmission of the acquired data.

The ESP32 microcontroller will act as a central hub, receiving the data streams from the ECG and PPG devices. It will process and prepare the data for transmission to the server

component. The connection between the ESP32 and the server will be depicted by an arrow, indicating the flow of data from the microcontroller to the server. The server component will be responsible for receiving and processing the data transmitted by the ESP32 microcontroller. It will perform various operations such as data storage, analysis, or integration with other systems. The processed data will then be presented to the user through the user interface component, which could be a graphical user interface (GUI), a web application, or any other suitable interface for displaying and interacting with the data.

3.1 Hardware Setup

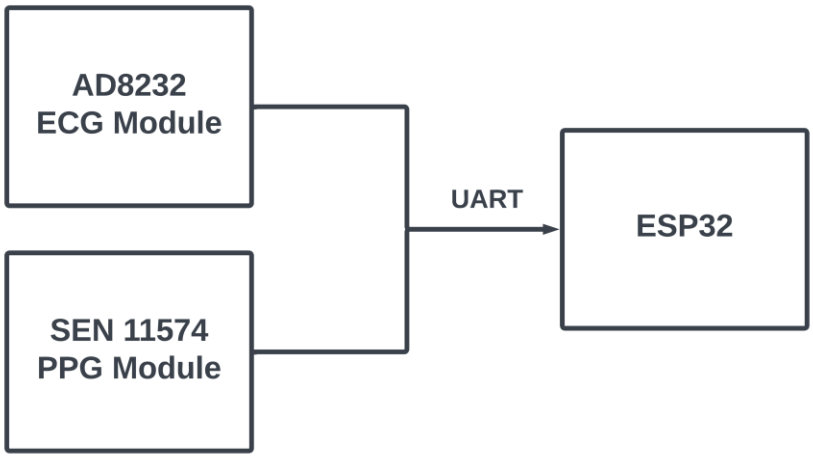


Figure 3.2: ECG and PPG Data transfer from sensors to ESP32 Microcontroller

As illustrated in Figure 3.2, the proposed system will capture physiological data using a combination of electrocardiogram (ECG) and photoplethysmogram (PPG) sensors. These sensors will capture electrical activity within the heart and variations in blood volume within the vasculature, respectively. The acquired signals will then be transmitted through a Universal Asynchronous Receiver/Transmitter (UART) communication protocol to an ESP32

3.1.1 Sensor Selection and Integration

AD8232 ECG Sensor: The AD8232 sensor was chosen for its exceptional performance in capturing Electrocardiogram (ECG) signals. This sensor accurately measures cardiac electrical activity, enabling the precise tracking of the heart's electrical impulses. SEN 11574 PPG Sensor: The SEN 11574 sensor is designed for Photoplethysmogram (PPG) signal acquisition. It excels in monitoring blood volume changes and pulse wave characteristics, providing a wealth of physiological information.

3.1.2 Microcontroller

ESP32-WROOM-32 is a powerful, generic Wi-Fi + Bluetooth® + Bluetooth LE MCU module that targets a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming and MP3 decoding. At the core of this module is the ESP32-D0WDQ6 chip*. The chip embedded is designed to be scalable and adaptive.

There are two CPU cores that can be individually controlled, and the CPU clock frequency is adjustable from 80 MHz to 240 MHz the chip also has a low-power coprocessor that can be used instead of the CPU to save power while performing tasks that do not require much computing power, such as monitoring of peripherals. ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, SD card interface, Ethernet, high-speed SPI, UART, I2S, and I2C.

3.2 Machine Learning Model

The methodology of developing and validating the Machine Learning model will be detailed below.

3.2.1 Selection of Machine Learning Models

This Study involves the application of various machine learning models such as LSTMs, RNNs and NARX to determine the optimal model for the prediction of Blood pressure. These models are distinguished by their ability to capture temporal dependencies within physiological time-series data, rendering them well-suited for the complex task of direct blood pressure estimation. The choice of these models is driven by their appropriateness for the rich and diverse data available in the MIMIC-III (Medical Information Mart for Intensive Care-III) dataset.

3.2.2 Feature Selection and Data Preprocessing

This approach places emphasis on thoughtful feature selection, given the extensive nature of the MIMIC-III dataset. Relevant features from the dataset are aligned with the underlying physiological context and the unique characteristics of the MIMIC-III data. In parallel with feature selection, data preprocessing procedures are vital. These include normalization and noise reduction techniques, which are essential to elevate the quality and integrity of our input data. The comprehensive nature of the MIMIC-III dataset emphasizes the significance of data preparation, ensuring the accuracy and robustness of the machine learning models.

3.2.3 Model Training and Validation

Both model training and validation will draw upon the MIMIC-III dataset. This dataset, comprising paired Electrocardiogram (ECG) and Photoplethysmogram (PPG) signals, alongside corresponding blood pressure measurements, will be partitioned into dedicated training and validation subsets. This partitioning approach is essential to provide an accurate reflection of model performance. The model training process will encompass the optimization of model parameters to facilitate the nuanced capture of the intricate relationships inherent in the MIMIC-III data. Through iterative refinement, our machine learning models will acquire the proficiency required for the generation of reliable, direct estimates of blood pressure.

3.2.4 Performance Evaluation

Performance evaluation within this approach will conform to rigorous standards. Standardized metrics, including mean absolute error (MAE), root mean square error (RMSE), and correlation coefficients, will serve as the quantitative metrics for assessing the accuracy and reliability of the blood pressure estimations generated by our machine learning models. dataset. Furthermore, the system will be deployed and tested on a group of 30 subjects to assess its real-world performance in comparison to a digital sphygmomanometer.

3.3 Server and User interface Setup

The system is divided into two main components: the frontend and the backend. Backend is based on the Flask framework. Flask, a microframework for Python, provides a compact foundation for constructing web applications. By utilizing routing, Flask effectively links URLs to functions, effectively managing user requests. Additionally, Flask seamlessly integrates with templating engines such as Jinja2, enabling the generation of dynamic HTML content. Furthermore, Flask supports extensions that enhance its functionality, including databases, user authentication, and object-relational mapping (ORM). The User Interface is designed using React JS. React JS, a JavaScript library intended for the creation of user interfaces, showcases its strength in producing dynamic and reusable UI components. Through the utilization of a virtual DOM (Document Object Model), React optimizes the rendering process by updating only the specific parts of the UI that have been modified. This virtual DOM greatly enhances the performance of web applications, especially those dealing with frequently changing data, as demonstrated in this system.

Once a patient is registered, the user can View/Record Patient Data, which allows them to access and update the patient's information. To view specific patient data, the user must Select Patient using the Patient ID. After selecting a patient, the user can then Select Sensor Data to View, which displays the sensor data collected for that particular patient. The frontend communicates with the backend through API calls, which facilitate the exchange of data between the two components. The backend component consists of several modules that

handle the server-side operations. The ESP 32 module, which is a microcontroller or a system-on-a-chip, sends HTTP requests to the Flask Server. The Flask Server is a web application framework written in Python, which acts as the central component of the backend.

The Flask Server interacts with two data sources: Patient Details and Patient Sensor Data. The Patient Details module stores and retrieves patient information, while the Patient Sensor Data module handles the sensor data collected for each patient. Both data sources are stored in an SQLite Database, which is a lightweight, file-based relational database management system.

Figure 3.3: Client Server communication for data processing and visualization

Figure 3.3 represents the software architecture of a patient monitoring system. The system is divided into two main components: the frontend and the backend. The frontend component represents the user interface and the interactions that a user can perform. It starts with a Home Page, which serves as the entry point to the application. From there, a user accesses the Patient Registration module, where they register new patients into the system.

The Figure 3.4 represents the process of recording and visualizing patient data using a cuffless blood pressure monitoring system and the associated user interface. The initial step involves the Patient component, where the user creates a user profile and provides their personal details. This information is then sent to the User Interface component. After receiving the patient details, the User Interface creates a request for a new patient ID to the Flask Server

component. The Flask Server generates a unique Patient ID and sends it back to the User Interface. It also creates the necessary tables (sensor values and average values) in the SQLite Database component to store the patient's data.

Once the patient is registered, they can log in to the system by providing their login credentials to the User Interface. The User Interface then verifies these credentials with the Flask Server. At this point, the patient can start measuring their blood pressure using the cuffless blood pressure monitoring system. The sensor data from the cuffless device is processed by the User Interface, which calculates the blood pressure values based on the incoming sensor data. The calculated blood pressure data is sent to the Flask Server for storage in the SQLite Database

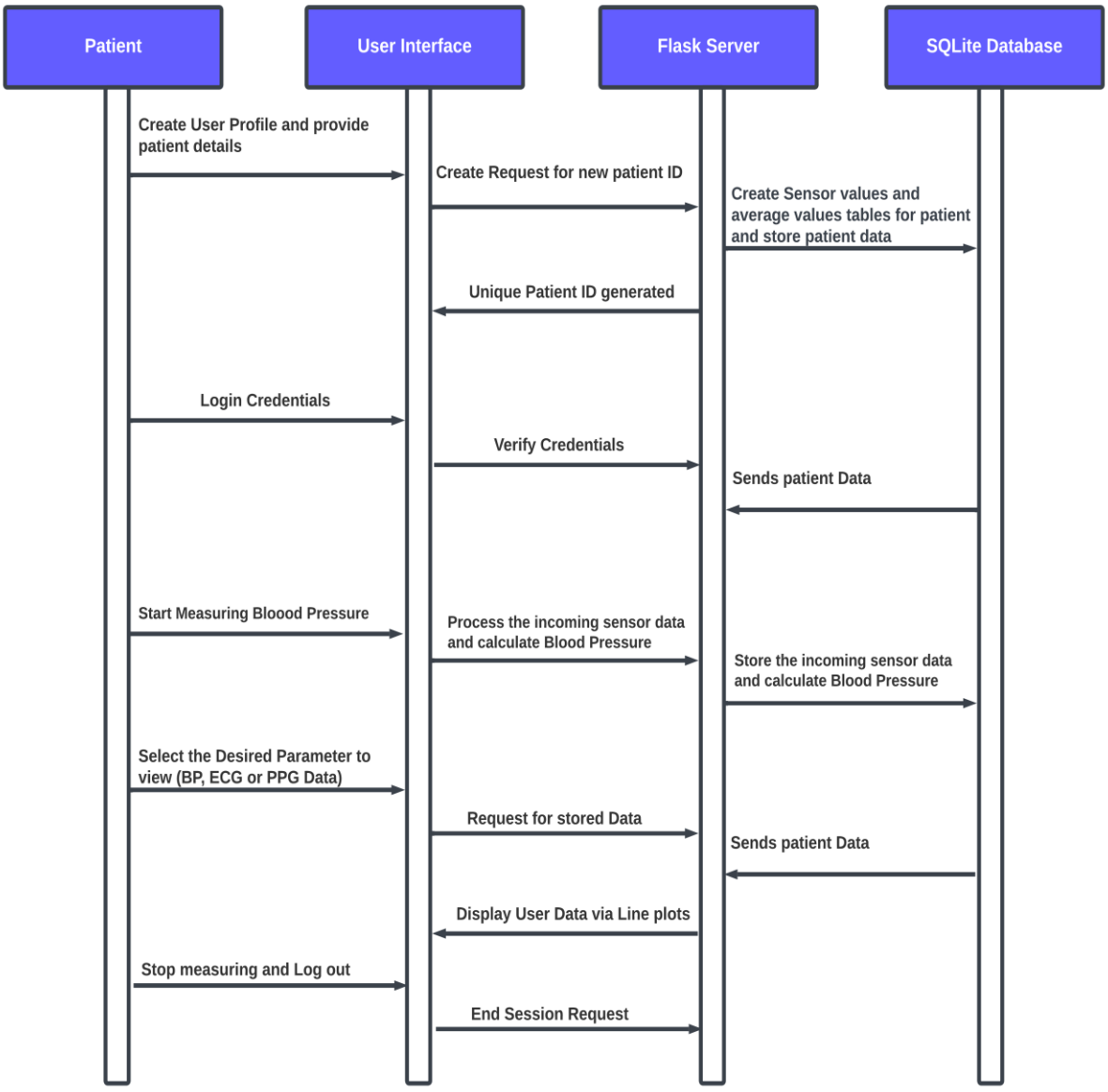


Figure 3.4: Process of Recording Blood Pressure via the Cuffless Blood Pressure Monitoring System by Patients

The Flask Server component receives the incoming sensor data, stores it in the appropriate tables, and calculates the blood pressure values based on the received data. After the data has been recorded, the patient can select the desired parameter to view, such as blood pressure (BP), electrocardiogram (ECG), or photoplethysmogram (PPG) data, through the User Interface. The User Interface then sends a request for the stored data to the Flask Server. The Flask Server retrieves the requested patient data from the SQLite Database and sends it back to the User Interface. The User Interface then displays the patient's data via line plots, allowing them to visualize and analyze their recorded data.

When the patient is done, they can stop measuring and log out by sending an end session request from the User Interface.This figure illustrates how the cuffless blood pressure monitoring system, the User Interface, the Flask Server, and the SQLite Database components work together to enable patients to record their blood pressure data, store it securely, and visualize it for analysis and monitoring purposes.

CHAPTER 4

IMPLEMENTATION AND RESULTS

In this section, the practical steps and procedures involved in the implementation of cuff-less blood pressure monitoring system using the approaches outlined in the preceding section is detailed along with the results of the implementation.

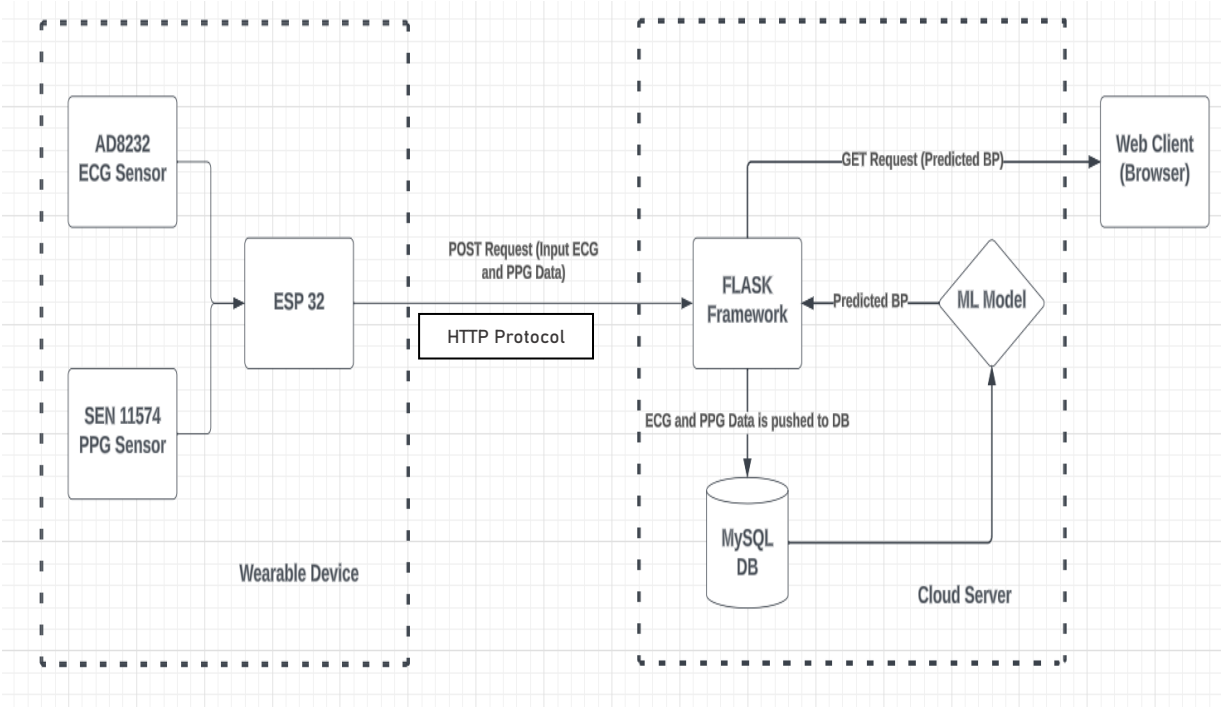


Figure 4.1: Data flow diagram of Cuffless Blood pressure monitoring system

4.1 Hardware Setup and Data Transfer to Server

The hardware components of the system have been configured to facilitate the collection, processing, and recording of physiological data. The ESP32 module connects to the local network using predefined SSID and password parameters. Subsequently, the module collects sensor data, including both ECG and PPG signals from the sensors, and packages it into JSON payloads. These payloads are then transmitted to the Flask server via HTTP POST requests. Additionally, the ESP32 periodically queries the server to ascertain the recording status, determining whether data collection should commence or cease based on the server's response.

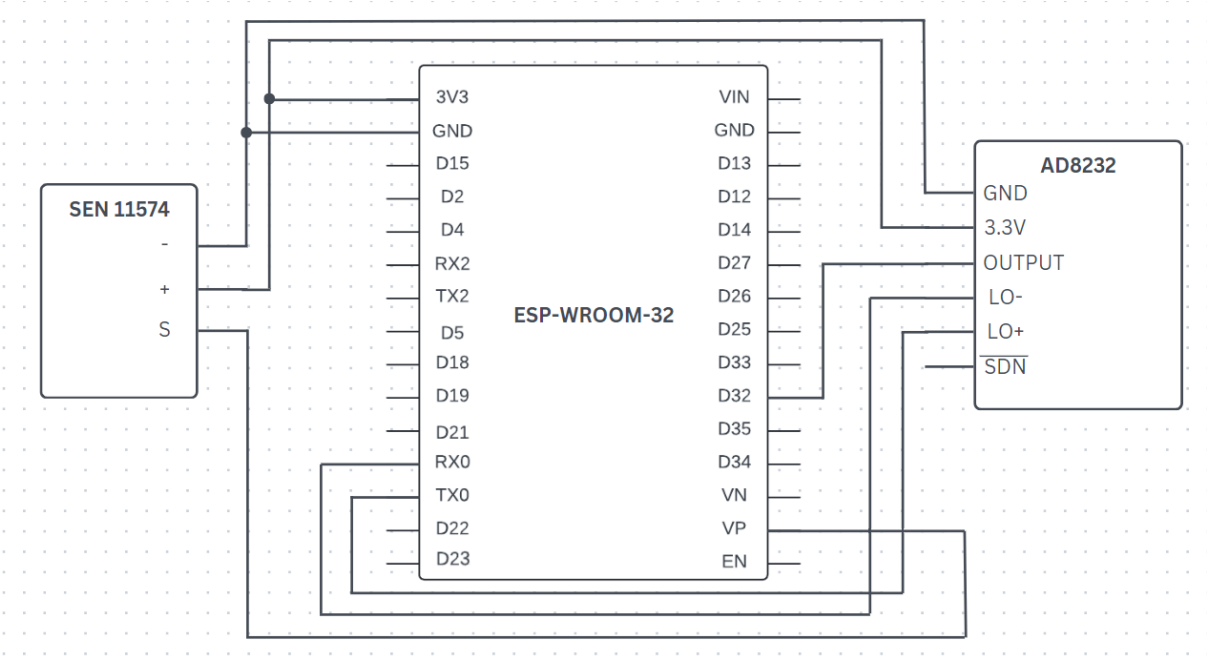


Figure 4.2: Schematic Diagram of ECG and PPG Signal Acquisition System

4.1.1 Backend Server – Flask Framework

The Flask backend is the central component of the Cuffless Blood Pressure Monitoring System, responsible for managing data storage, processing, and estimation tasks. It utilizes Flask, a lightweight web framework for Python, along with Flask-CORS extension to handle HTTP requests and enable Cross-Origin Resource Sharing. This allows seamless communication between the backend API and the frontend React application hosted on different origins.

A major aspect of the backend functionality lies in its interaction with the SQLite database named PPG_ECG_Data.db. This database serves as a repository for patient details, sensor data, and computed average blood pressure values. Tables within the database are structured to store patient details, sensor values (including ECG and PPG data), and the resulting average blood pressure values.

Endpoints within the backend API facilitate various functionalities, such as adding and editing patient details, verifying patient existence, and fetching sensor data for visualization. These endpoints handle incoming requests and execute corresponding actions, ensuring efficient data management and retrieval.

Machine learning techniques are integrated into the backend for blood pressure estimation. This involves leveraging a pre-trained LSTM model to process sensor data from

ECG and PPG sensors and estimate blood pressure values. Data preprocessing, model inference, and result interpretation are seamlessly integrated, allowing for accurate blood pressure estimation.

The backend also includes mechanisms for controlling measurement processes, monitoring status, and handling stoppage checks.

Endpoints are provided to start and stop measurement processes, along with functionality to monitor the status of ongoing measurements.

This ensures smooth operation and management of measurement processes, enhancing the overall reliability and usability of the system. Error handling mechanisms are implemented within the backend to gracefully manage exceptions and errors that may occur during data processing or database interactions.

4.1.2 Frontend Server – React JS Application

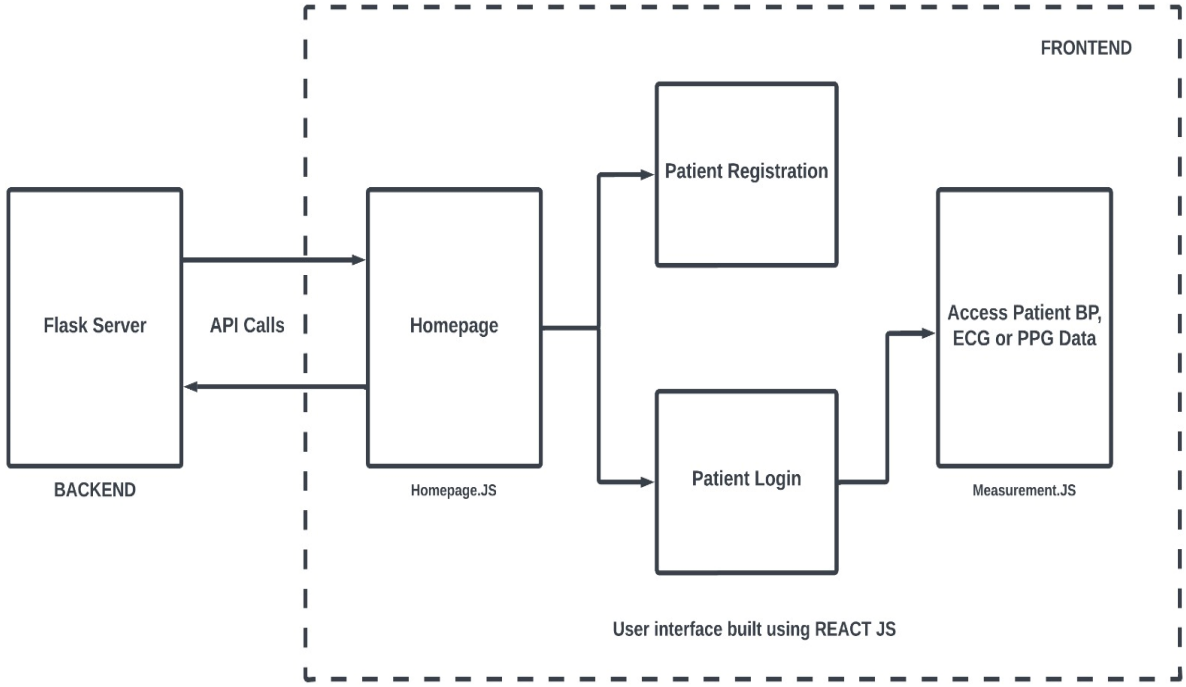


Figure 4.3: Data Flow Diagram of Server to User communication

The frontend React application is designed to facilitate patient management and measurement processes through a user-friendly interface.

The Home component serves as the landing page for the Cuffless Blood Pressure Monitoring System. It presents users with a welcoming message, "Welcome to Cuffless Blood Pressure Monitoring System", along with two buttons for navigating to different sections of the application.

The first button, labelled "Register Patient Details", redirects users to the patient registration form where they can input necessary information for registration.

The second button, labelled "View Patient Data", directs users to a page where they can view and manage patient data.

This simple and intuitive interface provides users with easy access to essential functionalities of the system, enhancing user experience and facilitating smooth navigation within the application.

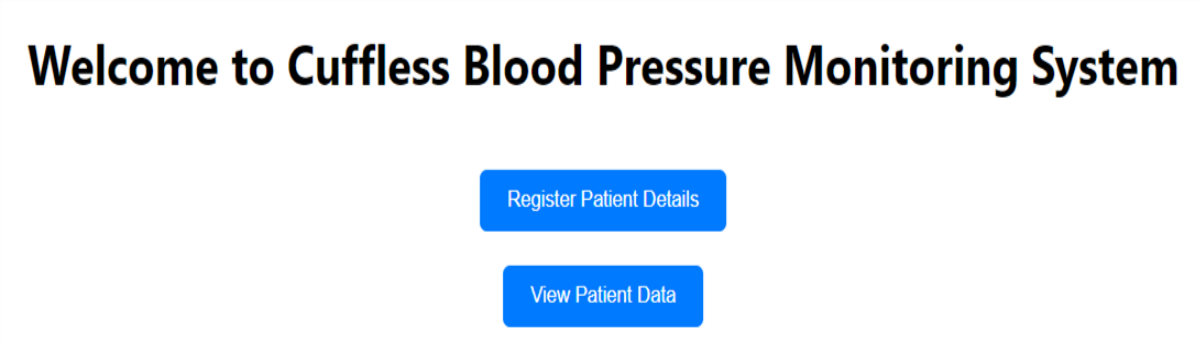


Figure 4.4: Home Screen for Registering the Details of New Patients and Viewing the Data of Old Patients

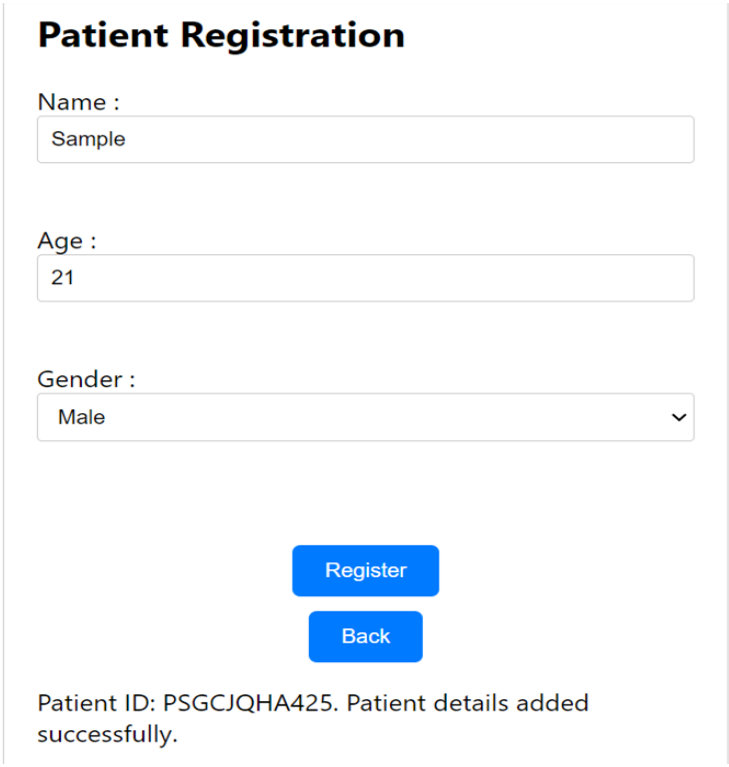


Figure 4.5: Successful Patient Registration with Creation of Unique Patient ID

The patient registration form, implemented in the PatientRegistration component, enables users to input essential details such as name, age, and gender. Upon submission, the form sends a POST request to the backend server at http://127.0.0.1:80/api/new_patient, containing the provided patient information.

The server processes the request and responds with a unique patient ID, which is displayed to the user as confirmation of successful registration. In case of errors, appropriate error messages are displayed to guide the user.

The SelectPatient component is responsible for allowing users to select a patient by entering their ID. Upon rendering, users are presented with a form where they can input a patient ID. The component utilizes React's useState hook to manage the state of the patientId input field and the verificationResult message.

When the form is submitted, the handleSubmit function is invoked. This function prevents the default form submission behavior and sends a POST request to the backend server at http://127.0.0.1:80/verify_patient, containing the entered patient ID. The server verifies the existence of the patient based on the provided ID.

If the server responds with a success status (200 OK), the component checks the response data. If the patient exists (data.exists is true), the user is redirected to the MeasurementComponent page corresponding to the selected patient ID using React Router's useNavigate hook. If the patient does not exist, an appropriate error message is displayed. In case of any errors during the verification process or network issues, appropriate error messages are displayed to the user.

The MeasurementComponent component serves as the interface for starting and stopping measurements for a specific patient as well as accessing patient sensor data. It dynamically updates based on the patient's ID obtained from the URL parameters.

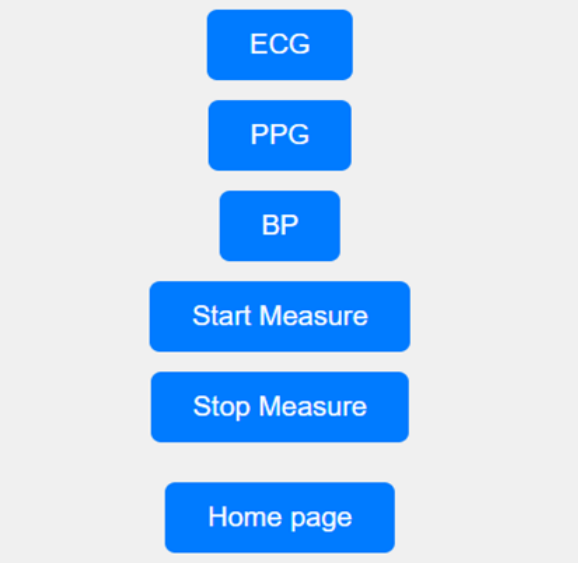


Figure 4.6: Data select screen to access patient data from database

Users can initiate measurements for various parameters such as ECG, PPG, and BP by clicking on the corresponding buttons.

Figure 4.7: Patient's PPG data accessed and plotted using the User Interface

When a measurement is initiated, the component sends HTTP requests to the backend server (http://127.0.0.1:80/measure) to begin the measurement process and update the patient's status (http://127.0.0.1:80/patient_id_api/${patientId}). Real-time feedback is provided to the user through status messages displayed on the interface.

Additionally, the application includes components for visualizing sensor data such as Electrocardiogram (ECG), Photoplethysmogram (PPG), and Blood Pressure (BP) data. For instance, the ECGData component fetches ECG data for the specified patient from the backend server (http://127.0.0.1:80/api/ecg/${patientId}) and renders it as a line chart using Chart.js.

This chart updates at regular intervals, providing real-time monitoring of the patient's ECG signals. Similarly, components for PPG and BP data visualization are structured similarly, ensuring consistency in functionality and user experience across different data types.

Overall, the transition to wireless data transfer and recording represents a significant advancement in the system's capabilities. Through seamless integration of hardware and software components, the system achieves efficient and reliable data transfer and recording, laying the foundation for comprehensive physiological signal analysis and medical research.

4.2 Execution of Proposed System

In this section, the algorithmic steps taken during the implementation phase to record data from the respective sensors, perform ECG and PPG signal filtering, and log the acquired physiological data is outlined.

4.2.1 Sensor Configuration and Setup

The first step in the data acquisition process revolves around the careful configuration and positioning of the AD8232 ECG sensor and the SEN 11574 PPG sensor. This step is pivotal in optimizing signal capture and ensuring that the sensors are primed for accurate physiological data acquisition.

AD8232 ECG Sensor Configuration:

Ensure precise positioning of the AD8232 ECG sensor to facilitate the accurate acquisition of electrocardiogram (ECG) signals. Establish a secure and consistent connection between the ECG electrodes and the subject's skin, adhering meticulously to standard electrode placement guidelines.

SEN 11574 PPG Sensor Setup:

Configure the SEN 11574 PPG sensor to capture photoplethysmogram (PPG) signals. Position the PPG sensor in a manner that optimizes its ability to monitor blood volume changes and pulse wave characteristics.

ESP32 Microcontroller Interface:

Utilize the ESP32 WROOM as the central hub for interfacing with the sensors.Implement code and connections that facilitate data transfer and synchronization between the sensors and the ESP32.

4.3 Machine Learning (ML) models

The goal of this project is to develop machine learning models that can estimate Blood Pressure (BP) from Electrocardiogram (ECG) and Photoplethysmogram (PPG) signals. The model training uses the MIMIC III dataset, which provides ECG, PPG, and invasive arterial blood pressure (ABP) data.

Dataset Information

- Dataset: MIMIC III Dataset - Version 1.4
- Sampling Rate: 125 Hz
- Entities:
    - ECG: 61,000 samples, PPG: 61,000 samples
    - BP (Invasive Arterial Blood Pressure): 61,000 samples
- Train-Test Split: 80% - 20%

Models

Four different machine learning models were implemented and evaluated for this task.

4.3.1 Model 1: Deep Neural Network (DNN)

Architecture:

- Input Layer: 64 neurons with ReLU activation function
- Hidden Layer: 64 neurons with ReLU activation function
- Output Layer: 1 neuron for BP estimation

Loss Function: Mean Squared Error (MSE)

Optimizer: Adam optimizer with default learning rate

Training Details:

- Training Data: ECG and PPG sensor data
- Epochs: 100, Batch Size: 32

- Evaluation Metric: Root Mean Squared Error (RMSE)

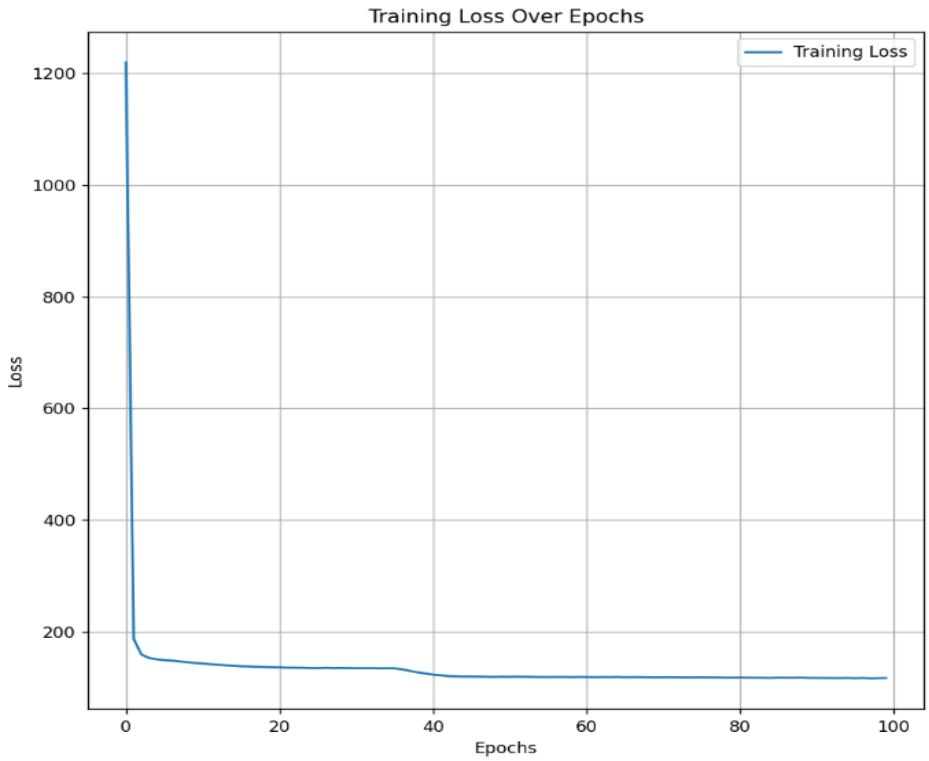


Figure 4.8: Training loss over Epochs of DNN

The plot shows that the loss decreases sharply at the beginning, indicating rapid learning or improvement in the model's performance. After this initial phase, the decrease in loss slows down, suggesting that as the epochs increase, the amount by which the model omproves lessens. After around 10 epochs, the line begins to flatten out significantly, indicating a slower rate of decrease in training loss. The overall trend of the graph suggests an initial rapid improvement, followed by a gradual approach towards minimal change.

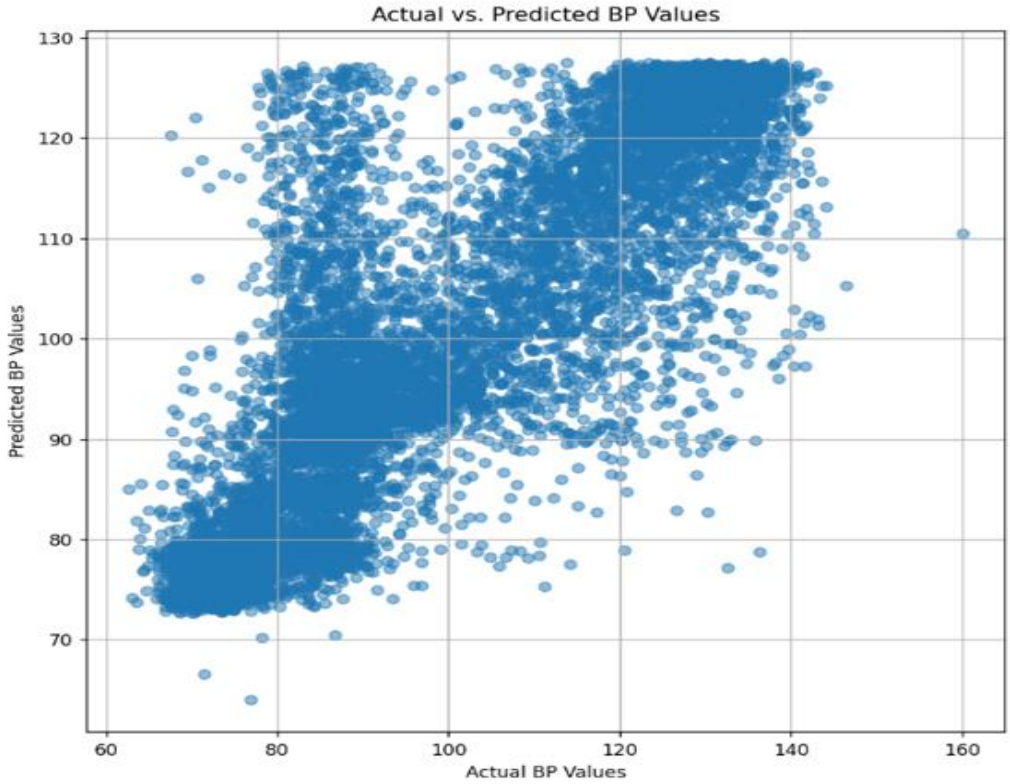


Figure 4.9: Comparison between Actual and Predicted BP values of DNN

RMSE: 10.5

The graph illustrates the relationship between actual blood pressure (BP) values and their corresponding predicted values. A positive correlation is evident, indicating that as actual BP values increase, the predicted BP values also tend to increase. Most data points cluster along the diagonal line, suggesting accurate predictions for the majority of cases. However, there are outliers where the predicted values significantly deviate from the actual ones, warranting further investigation.

4.3.2 Model 2: Long Short-Term Memory (LSTM) Network

Architecture:

- Input Layer: ECG and PPG input feature with LSTM units.
- LSTM Layer 1: 64 LSTM units.
- LSTM Layer 2: 64 LSTM units.
- Output Layer: 1 neuron for BP estimation.
- Loss Function: Mean Squared Error (MSE).
- Optimizer: Adam optimizer with default learning rate.

Hyperparameter Tuning with Random Search:

- Hyperparameters Searched: Number of LSTM units in each layer (units).
- Hyperparameter Space: Min value = 32, Max value = 256, Step size = 32.
- Objective: Minimization of validation loss.

- Max Trials: 5.
- Execution per Trial: 1.

Training Details:

- Training Data: ECG and PPG sensor data.
- Epochs: 79 epochs (Early Stopping)
- Batch Size: 32.
- Evaluation Metric: Root Mean Squared Error (RMSE) , Mean Absolute Error (MAE)

ML Model Architecture

The ML Model Architecture is generated using Keras plot_model function , There are 3 Layers (2 LSTM , 1 Dense) . The input shape of LSTM layer is of the form (batch size, timestep , features) , None in batch size field specifies that it can accomodate any no. of rows (data points) , timestep is 1 as default (use only the current data point for estimation) and features are 2 corresponding to ECG and PPG data. The output of the First LSTM layer has 256 features that is given as input to the next LSTM layer , which then produces an output of 256 features. Finally in the Dense layer , the features are combined and a single estimated Blood Pressure value is given as output.

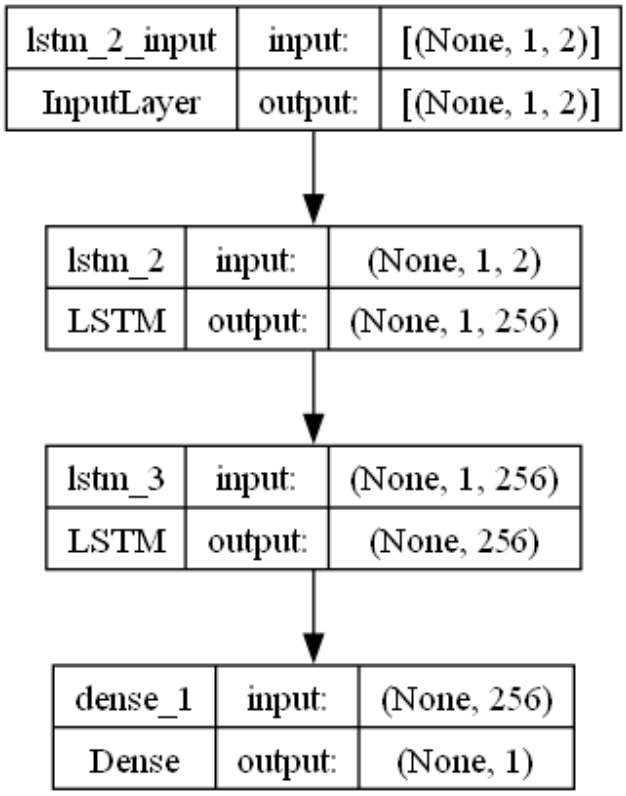


Figure 4.10: LSTM Model Architecture

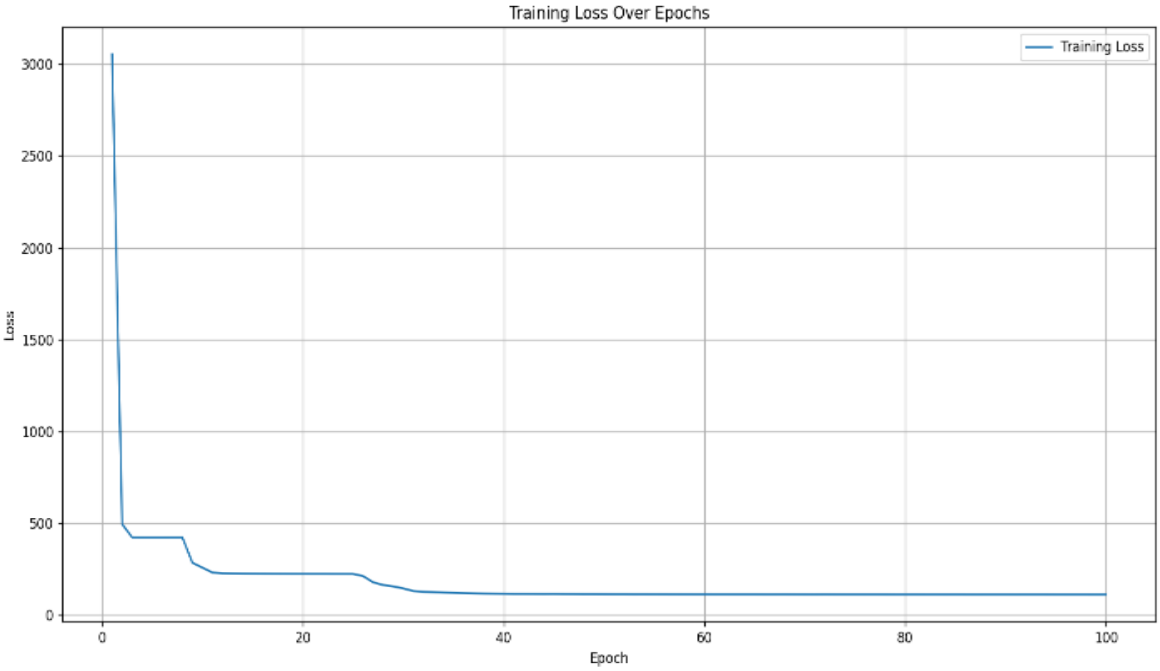
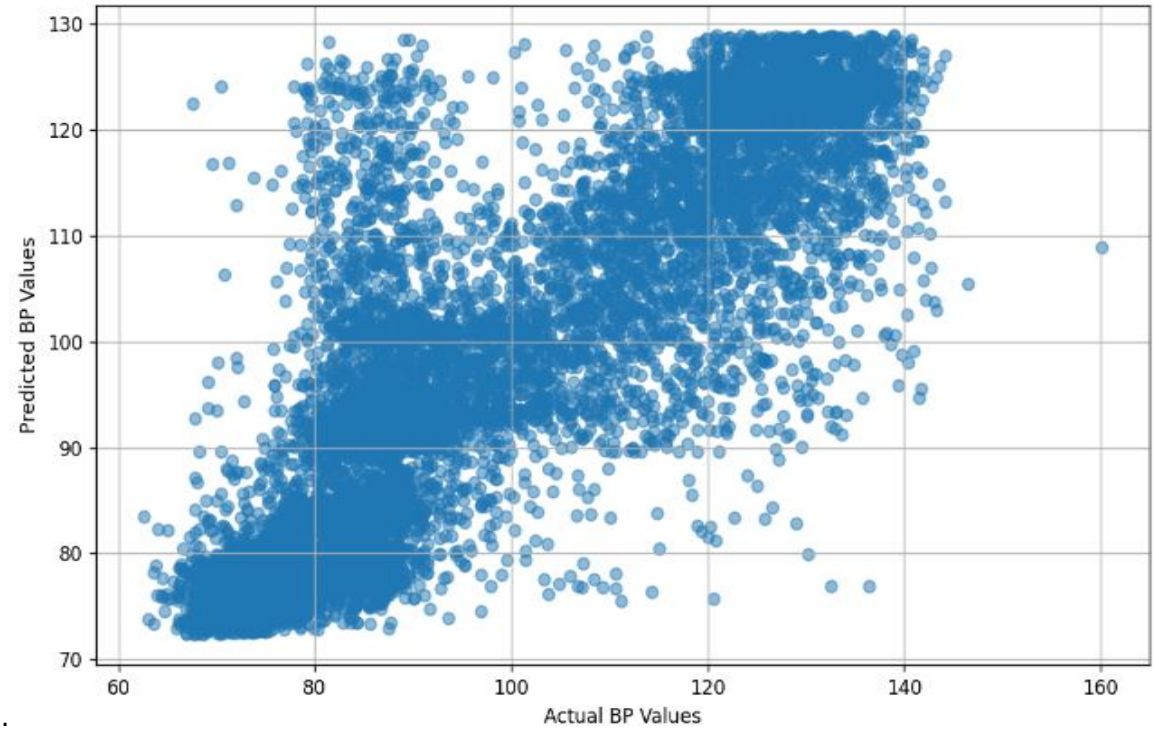


Figure 4.11: Training loss over Epochs of LSTM

The graph shows a sharp decline in loss at the beginning, which gradually levels off as the number of epochs increases, indicating that the model is learning and improving its predictions over time.

The x-axis represents the number of training epochs, while the y-axis represents the loss value. Initially, the loss decreases rapidly, suggesting that the model is quickly adapting to the training data. However, after a certain point, the decrease in loss becomes less pronounced, indicating that further training may not significantly improve performance.



.

Figure 4.12: Scatter Plot between Actual and Predicted BP values of LSTM

RMSE: 10.3

The points on the scatter plot seem to cluster around a diagonal line, suggesting a positive correlation between actual and predicted BP values. While the model generally performs well in predicting BP values within a certain range, there are instances where it deviates from the ideal line. Some data points lie farther away from the main cluster, indicating outliers. These outliers could be due to measurement errors, anomalies in the dataset, or limitations of the model.

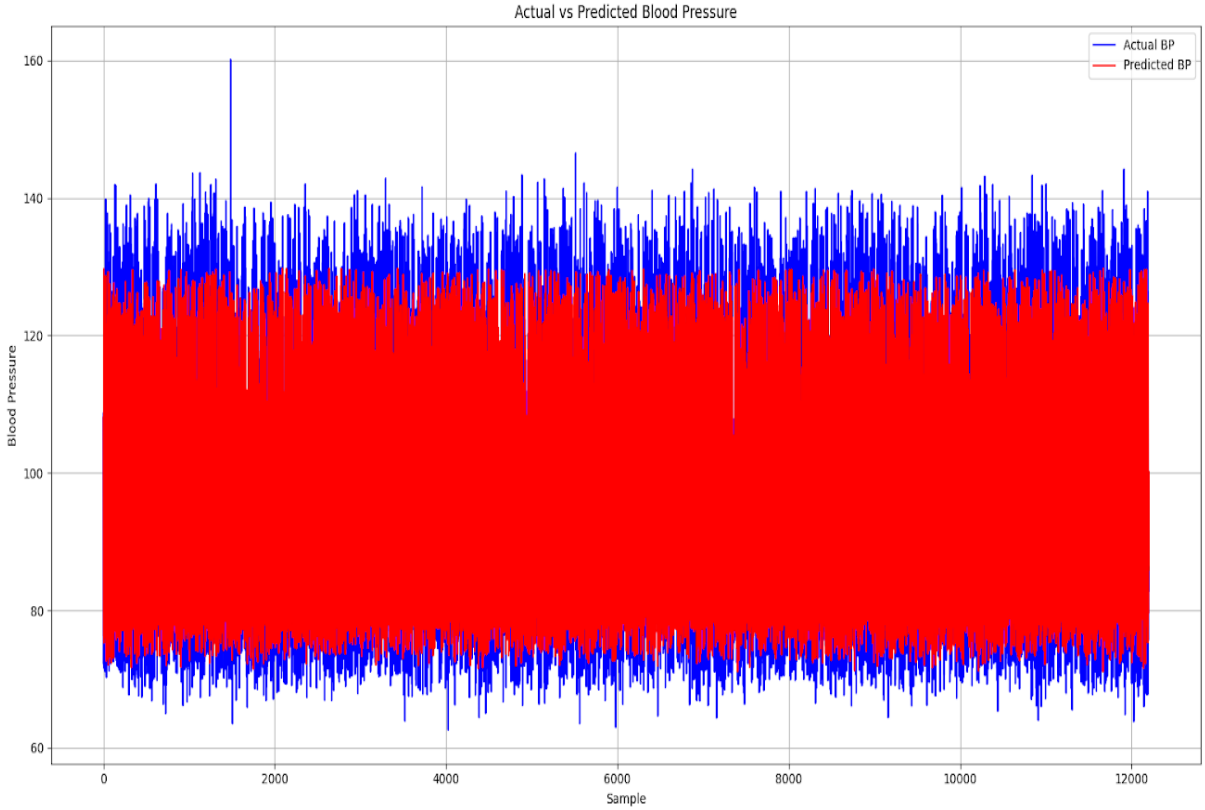


Figure 4.13: Comparison of Actual and Predicted Blood Pressure by LSTM Network

MAE: 7.02

The graph depicts two sets of data: actual blood pressure (in blue) and predicted blood pressure (in red) across a range of samples. The data shows that both the actual and predicted blood pressure values exhibit fluctuations throughout the sample range. The two lines (blue for actual and red for predicted) follow a similar pattern, suggesting that the predictions align closely with the actual measurements. The sample range goes from 0 to approximately 12,000 samples, and systolic blood pressure values range from around 60 mmHg to just over 200 mmHg.

4.3.3 Model 3: Recurrent Neural Network (RNN)

Architecture:

- Input Layer: 64 neurons with ReLU activation function
- Hidden Layer: 64 neurons with ReLU activation function
- Output Layer: 1 neuron for BP estimation

Loss Function: Mean Squared Error (MSE)

Optimizer: Adam optimizer with default learning rate

Training Details:

- Training Data: ECG and PPG sensor data
- Epochs: 100, Batch Size: 32

Evaluation Metric: Root Mean Squared Error (RMSE)

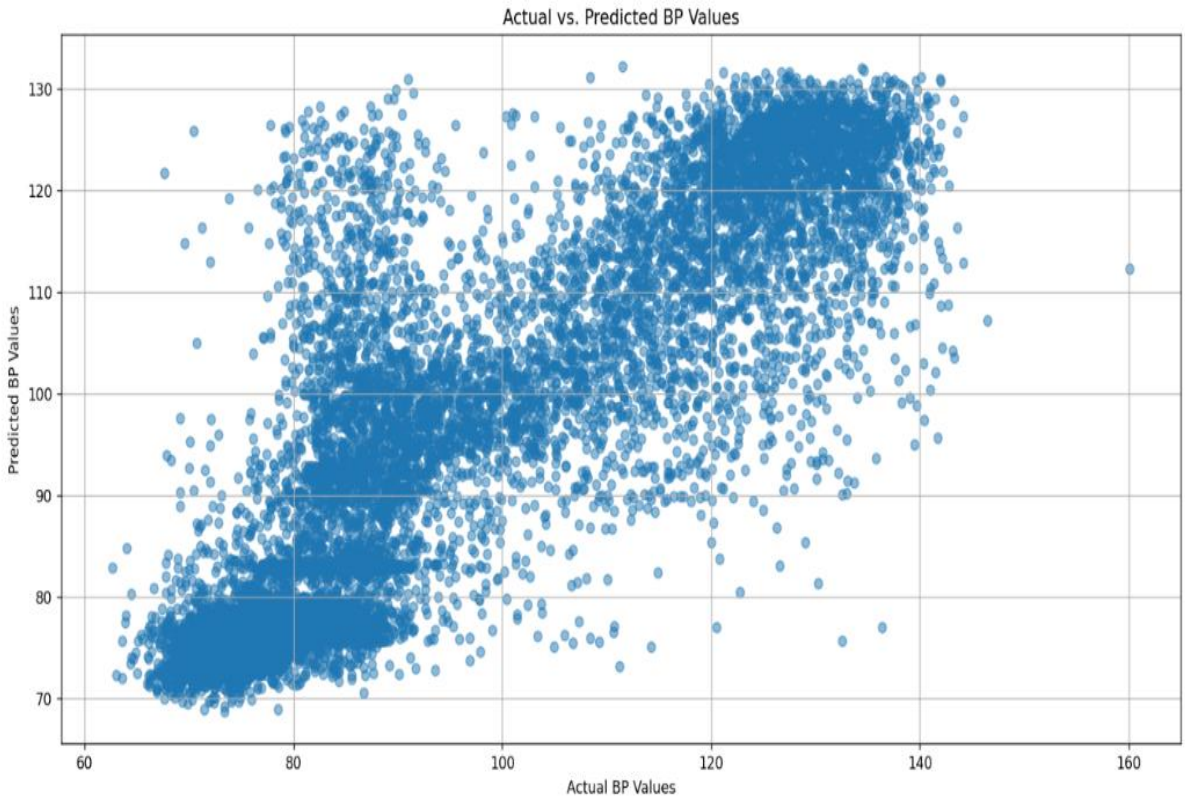


Figure 4.14: Scatter Plot between Actual and Predicted BP values of RNN

There is no clear linear relationship visible between actual and predicted BP values. Instead, we see a dense cloud of data points around the centre, with some spread along both axes. The lack of a distinct pattern suggests that the prediction model may not be capturing the underlying relationship effectively.

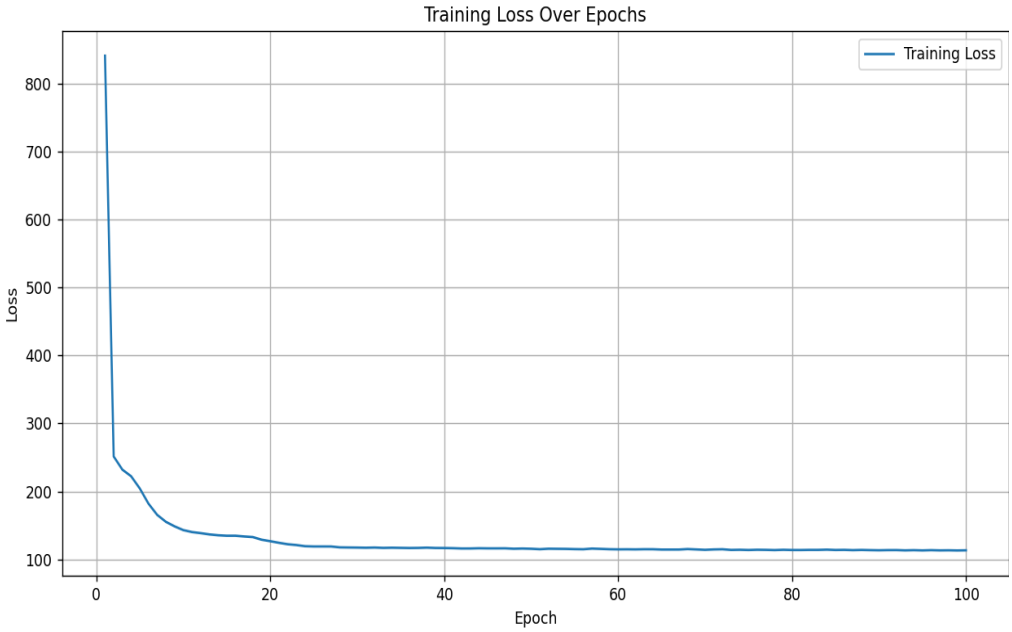


Figure 4.15: Training loss over Epochs of RNN

RMSE : 10.44

The steep initial decline suggests that the model is learning effectively. The plateau indicates that further training may not significantly improve performance. It's essential to monitor validation loss and avoid overfitting.

4.3.4 Model 4: Nonlinear Autoregressive network with exogenous inputs (NARX)

Architecture

Input Layer:

- Total 9 input features in input layer

Hidden Layers:

- First dense layer: 32 units, ReLU activation, with L2 regularization to prevent overfitting by shrinking weights.
- Dropout layer: Rate of 0.2, applied after the first hidden layer.
- Second dense layer: 64 units, ReLU activation, with L2 regularization applied, followed by another dropout layer with a rate of 0.2.

Output Layer:

- Dense layer with a single unit.

Training Details:

- Training Data: ECG and PPG sensor data
- Epochs: 382, Batch Size: 32

Evaluation Metrics:

MAE (Mean Absolute Error) = 13.16. The MAE represents the average absolute difference between the estimation and true blood pressure values. This value is significantly higher than our LSTM model which has an MAE value of 7.02.

RMSE (Root Mean Squared Error) = 13.65. This value is higher than our LSTM Model which has an RMSE of 10.33.

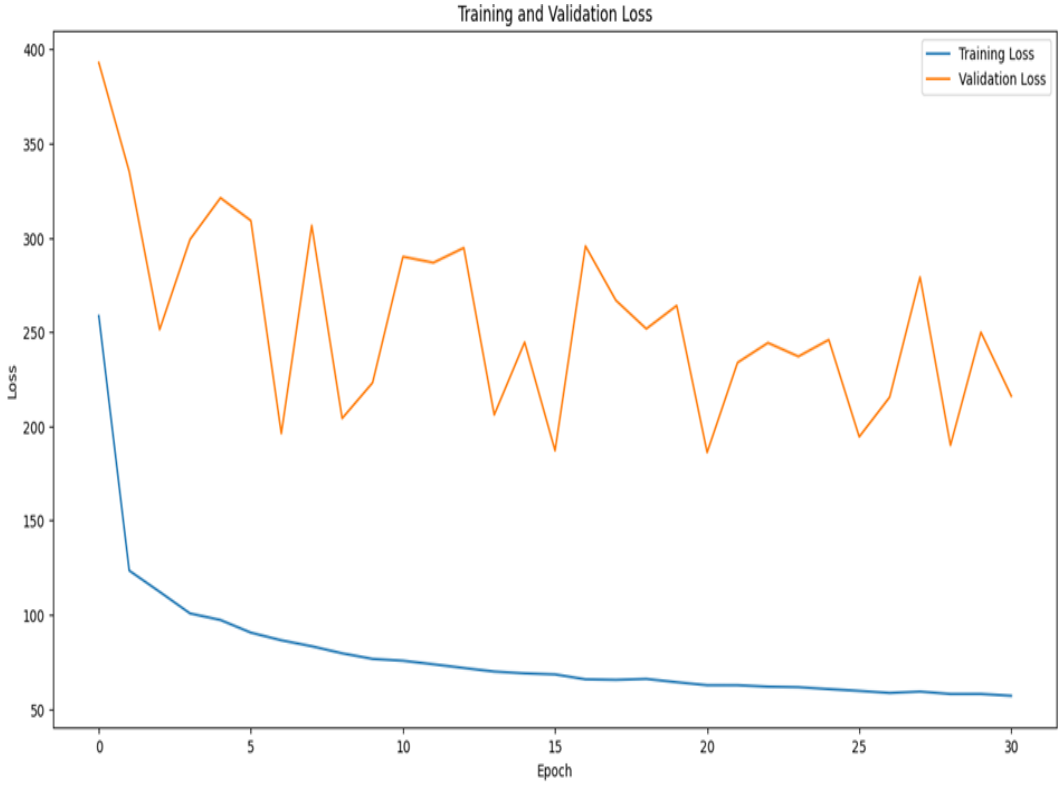


Figure 4.16: Training loss over validation of NARX

Figure 4.17: Comparison between Predicted and True BP of NARX

The spread of data points suggests varying degrees of accuracy in the prediction model. Ideally, the points should align along a straight line (y = x), indicating perfect predictions. Deviations from this diagonal line indicate model errors or biases. The dense clustering of points around the diagonal suggests reasonable predictions.

4.3.5 Testing with Real-Time data:

RNN, LSTM and DNN models provide similar RMSE values. The LSTM Network, represented by Model 2, presents several distinct advantages in this context.

Firstly, it excels in handling temporal dependencies within time-series data, a critical feature for accurately estimating blood pressure from ECG and PPG signals. Its specialized architecture allows it to effectively capture intricate patterns and relationships in sequential data, making it particularly valuable when dealing with complex physiological signals.

Additionally, LSTMs demonstrate robustness in scenarios where data is irregularly sampled, a feature that standard feedforward neural networks may struggle with. Moreover, the LSTM's ability to retain memory of past information over longer sequences proves invaluable in tasks where information from earlier points is pivotal for accurate estimations.

Real-time ECG and PPG data were collected using the AD8232 and Pulse Sensor over a period of 2 minutes. The collected data was then fed into a Long Short-Term Memory (LSTM) neural network for blood pressure estimation. The LSTM model was trained to estimate blood pressure ranges, and the results obtained fall within the expected values.

The following steps were performed to estimate blood pressure using the LSTM model:

1. **Data Reception:** The Flask server receives POST requests containing JSON payloads containing ECG and PPG sensor data from the ESP32 device. This data is sent to the /send_data endpoint.
2. **Data Processing:** Upon receiving the sensor data, the Flask server extracts the ECG and PPG data from the JSON payload. The received data is pre-processed, including scaling to ensure consistency with the training data used to train the LSTM model.
3. **Model Inference:** Once the data is pre-processed, it is reshaped to match the input requirements of the LSTM model. The reshaped data is then passed to the LSTM model for inference. The model estimates blood pressure based on the provided ECG and PPG data.
4. **Peak and Valley Detection:** After estimating blood pressure, the Flask server may perform additional processing, such as peak and valley detection, to refine the blood pressure estimations. This step might involve identifying systolic peaks and diastolic valleys within the estimated blood pressure values.
5. **Database Storage:** The estimated blood pressure values, along with any additional processed data, are stored in the SQLite database. This ensures that the estimations are logged for future reference and analysis.
6. **Response:** Finally, the Flask server sends a response back to the ESP32 device, indicating the success of the data processing and estimation steps.

This response may include any relevant information, such as confirmation of successful estimation or any errors encountered during the process.

4.4. Arterial (Systolic and Diastolic) BP Estimation Algorithm

Once the LSTM model estimates the BP values, these values are fed into algorithms that further estimates the Systolic and Diastolic BP values from the given BP array. Three methods were developed and tested on estimated noisy (with sudden spikes & dips) and noiseless BP array to find out the optimal method to be adopted.

Input 1:

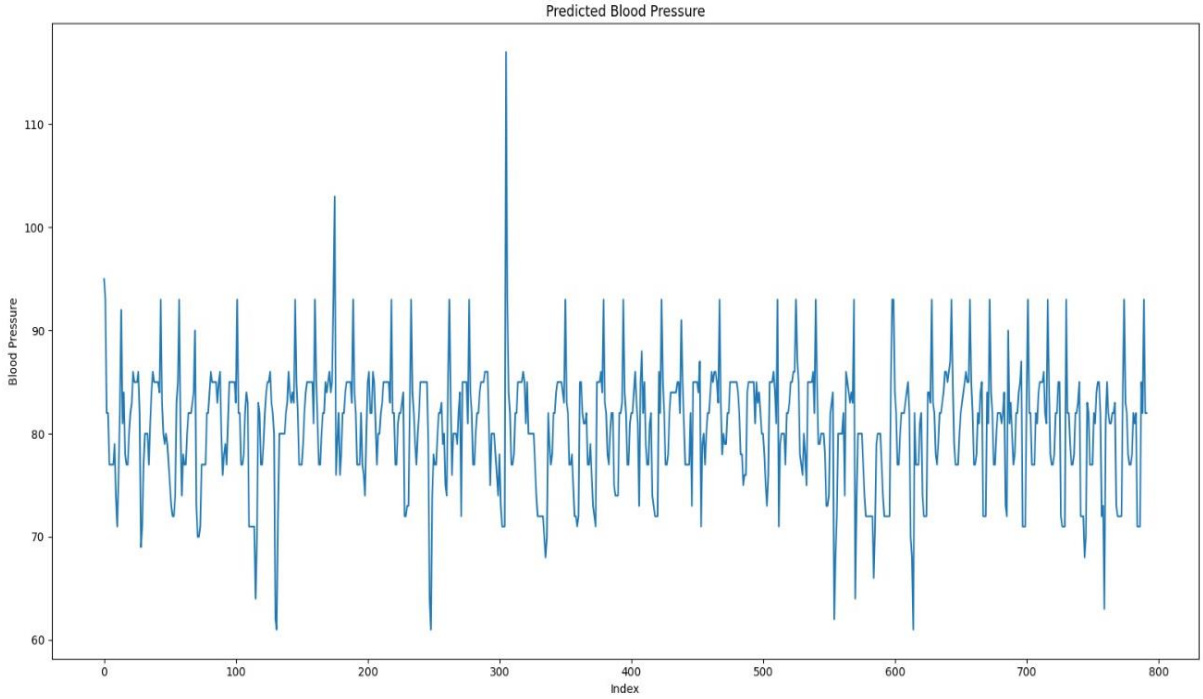


Figure 4.18: Estimated Noisy BP Array. Actual BP: (93/62)

Input 2:

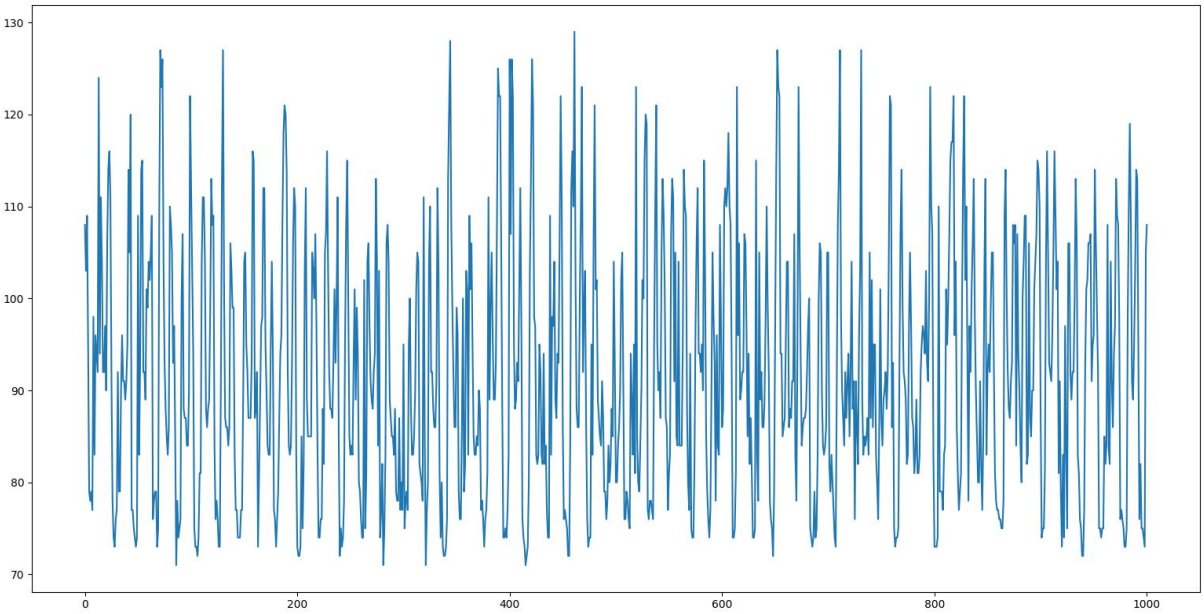


Figure 4.19: Estimated Noiseless BP Array. Actual BP: (120/71)

Method 1:

1. Find the max amp in BP array
2. Find the peaks that are b/w max val. and max val. - 10
3. Find local minima (valleys) when a point is less than its neighbours
4. Sys. BP = mean (peaks)
5. Dia. BP = mean (valleys)

Output:

Noisy BP array:

```
Avg Systolic BP:  117
Avg Diastolic BP:  76
```

Noiseless BP array:

```
Avg Systolic BP:  123
Avg Diastolic BP:  83
```

Method 2:

The problem with method 1 was when there is a sudden spike or sudden dip in the predicted BP value due to noise, the Systolic BP becomes incorrect. Ex: 117 is a single spiked value due to noise that caused the Systolic BP value to inflate to 117 as no value is b/w 117 and 107.

So, in method 2, only the values b/w 1st percentile and 99th percentile are taken into account, thus eliminating the sudden spikes and dips (outliners).

1. Find the 1st percentile and 99th percentile value in the predicted BP array
2. Any values below the 1st percentile will be set to the value of the 1st percentile
3. Any values above the 99th percentile will be set to the value of the 99th percentile
4. Find the max amp in BP array
5. Find the peaks that are b/w max val. and max val. - 5
6. When no other sys peak is found, Sys BP = max val.
7. Find local minima (valleys) when a point is less than its neighbours
8. Sys. BP = mean (peaks)
9. Dia. BP = mean (valleys)

Output:

Noisy BP:

```
Avg Systolic BP:  93
Avg Diastolic BP:  77
```

Noiseless BP:

```
Avg Systolic BP:  124
Avg Diastolic BP:  84
```

Method 2 gives more accurate result than the method 1 for noisy BP signal.

Method 3:

Systolic BP:

1. Find the max val. in BP array
2. Count no. of val. b/w max val. to max val. - 7
3. If Count > 3:
   Sys. BP = mean (val. b/w max val. to max val. - 7)

Else:
Move to next Peak (2nd max val.) and repeat the process

Diastolic BP:

1. Find the min val. in BP array
2. Count no. of val. b/w min val. to min val. + 7
3. If Count > 3:
   Dia. BP = mean (val. b/w min val. to min val. + 7)
   Else:
   Move to next Valley (2nd mix val.) and repeat the process

Output:

Noisy BP:

```
Avg Systolic BP:  92
Avg Diastolic BP:  65
```

Noiseless BP:

```
Avg Systolic BP:  124
Avg Diastolic BP:  75
```

Method 1 gives inaccurate results when there is noise in BP array, but does better Systolic BP estimation for noiseless BP array. While Method 2 performs better for noisy BP array. Both Method 1 and 2 are quite inaccurate in Diastolic BP estimation. Method 3 is also resistant to noise in BP array and gives more accurate Diastolic BP estimation that is close to the actual value.

4.5 Database Architecture

1. **Database File:**

   The database file is named "PPG_ECG_Data.db".

2. **Tables:**

Name	Type	Schema
Tables (14)		
average_values_PSG1ZILO14Z		CREATE TABLE average_values_PSG1ZILO14Z (id INTEGER PRIMARY KEY AUTOINCREMENT, average_diastolic INTEGER, average_systolic INTEGER)
average_values_PSG6H9DY7XL		CREATE TABLE average_values_PSG6H9DY7XL (id INTEGER PRIMARY KEY AUTOINCREMENT, average_diastolic INTEGER, average_systolic INTEGER)
average_values_PSG98HKQUYY		CREATE TABLE average_values_PSG98HKQUYY (id INTEGER PRIMARY KEY AUTOINCREMENT, average_diastolic INTEGER, average_systolic INTEGER)
average_values_PSGCJQHA425		CREATE TABLE average_values_PSGCJQHA425 (id INTEGER PRIMARY KEY AUTOINCREMENT, average_diastolic INTEGER, average_systolic INTEGER)
average_values_PSGJTADYG62		CREATE TABLE average_values_PSGJTADYG62 (id INTEGER PRIMARY KEY AUTOINCREMENT, average_diastolic INTEGER, average_systolic INTEGER)
average_values_PSGNTQABW7I		CREATE TABLE average_values_PSGNTQABW7I (id INTEGER PRIMARY KEY AUTOINCREMENT, average_diastolic INTEGER, average_systolic INTEGER)
patient_details		CREATE TABLE patient_details (patient_id TEXT PRIMARY KEY, name TEXT, age INTEGER, gender TEXT)
sensor_values_PSG1ZILO14Z		CREATE TABLE sensor_values_PSG1ZILO14Z (id INTEGER PRIMARY KEY AUTOINCREMENT, ecg_value INTEGER, ppg_value INTEGER, predicted_bp INTEGER)
sensor_values_PSG6H9DY7XL		CREATE TABLE sensor_values_PSG6H9DY7XL (id INTEGER PRIMARY KEY AUTOINCREMENT, ecg_value INTEGER, ppg_value INTEGER, predicted_bp INTEGER)
sensor_values_PSG98HKQUYY		CREATE TABLE sensor_values_PSG98HKQUYY (id INTEGER PRIMARY KEY AUTOINCREMENT, ecg_value INTEGER, ppg_value INTEGER, predicted_bp INTEGER)
sensor_values_PSGCJQHA425		CREATE TABLE sensor_values_PSGCJQHA425 (id INTEGER PRIMARY KEY AUTOINCREMENT, ecg_value INTEGER, ppg_value INTEGER, predicted_bp INTEGER)
sensor_values_PSGJTADYG62		CREATE TABLE sensor_values_PSGJTADYG62 (id INTEGER PRIMARY KEY AUTOINCREMENT, ecg_value INTEGER, ppg_value INTEGER, predicted_bp INTEGER)
sensor_values_PSGNTQABW7I		CREATE TABLE sensor_values_PSGNTQABW7I (id INTEGER PRIMARY KEY AUTOINCREMENT, ecg_value INTEGER, ppg_value INTEGER, predicted_bp INTEGER)
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)

Table 4.1: List of Tables in Database

- **patient_details:** This table stores patient-specific information, including patient ID, name, age, and gender.

	patient_id	name	age	gender
	Filter	Filter	Filter	Filter
1	PSG6H9DY7XL	Kamal	21	male
2	PSGNTQABW7I	Shanmugaraja T	45	male
3	PSGJTADYG62	Sample	12	other

Table 4.2: List of Patients in Database

- **sensor_values_{patient_id}:** Dynamic tables are created for each patient to store sensor readings. These tables include columns for ECG values, PPG values, and estimated blood pressure.

id	ecg_value	ppg_value	predicted_bp
Filter	Filter	Filter	Filter
1	1734	1664	89
2	1712	2288	83
3	1777	2867	78
4	1844	3818	76
5	1951	4095	75

Table 4.3: Patient Sensor Data

- **average_values_{patient_id}:** Similar to sensor_values tables, dynamic tables are created for each patient to store average blood pressure values. These tables typically contain columns for average systolic and diastolic blood pressure.

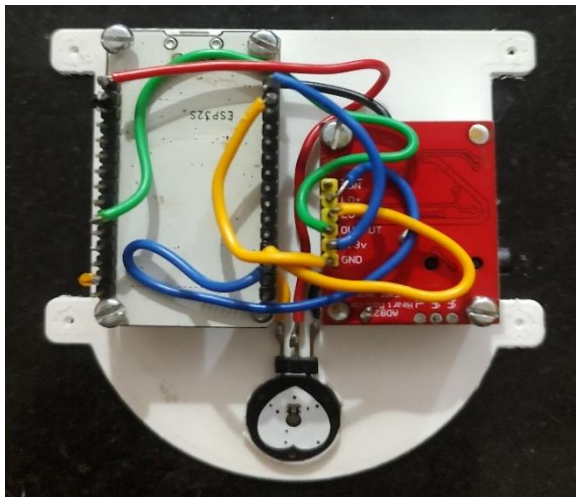
id	average_diastolic	average_systolic
Filter	Filter	Filter
1	81	126
2	80	112
3	78	121
4	87	122
5	81	129

Table 4.4: Calculated Average Blood Pressure Data

3. **Table Relationships:**
   - There's a one-to-many relationship between the patient_details table and the sensor_values_{patient_id} and average_values_{patient_id} tables.
   - Each patient in the patient_details table can have multiple entries in the sensor_values and average_values tables, corresponding to different sensor readings and average blood pressure values, respectively.

4. **Database Operations:**
    - Data insertion: Data from the ESP32 device, including ECG, PPG, and estimated blood pressure readings, are inserted into the respective sensor_values_{patient_id} table.
    - Data retrieval: Patient-specific data, such as sensor readings and average blood pressure values, can be retrieved from the appropriate tables for further analysis or display on the user interface.
5. **Data Integrity:**
    - Primary keys: Each table likely has a primary key column (e.g., id) to ensure unique identification of records.

4.6 Data Acquisition Module



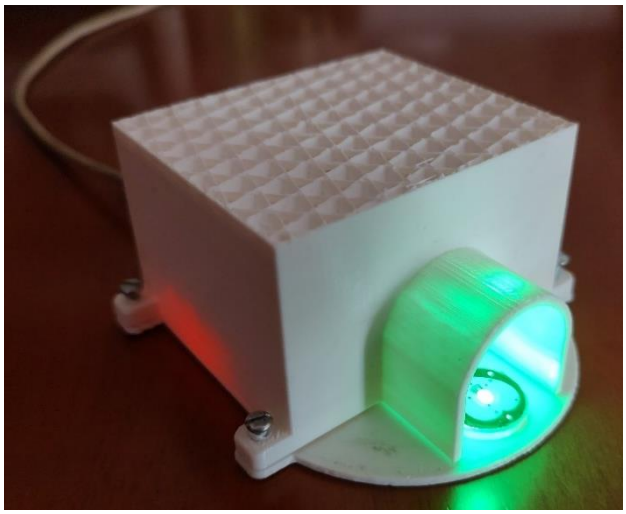


Figure 4.20: Hardware Interconnections and the Assembled Data Acquisition Module

Estimated Blood Pressure

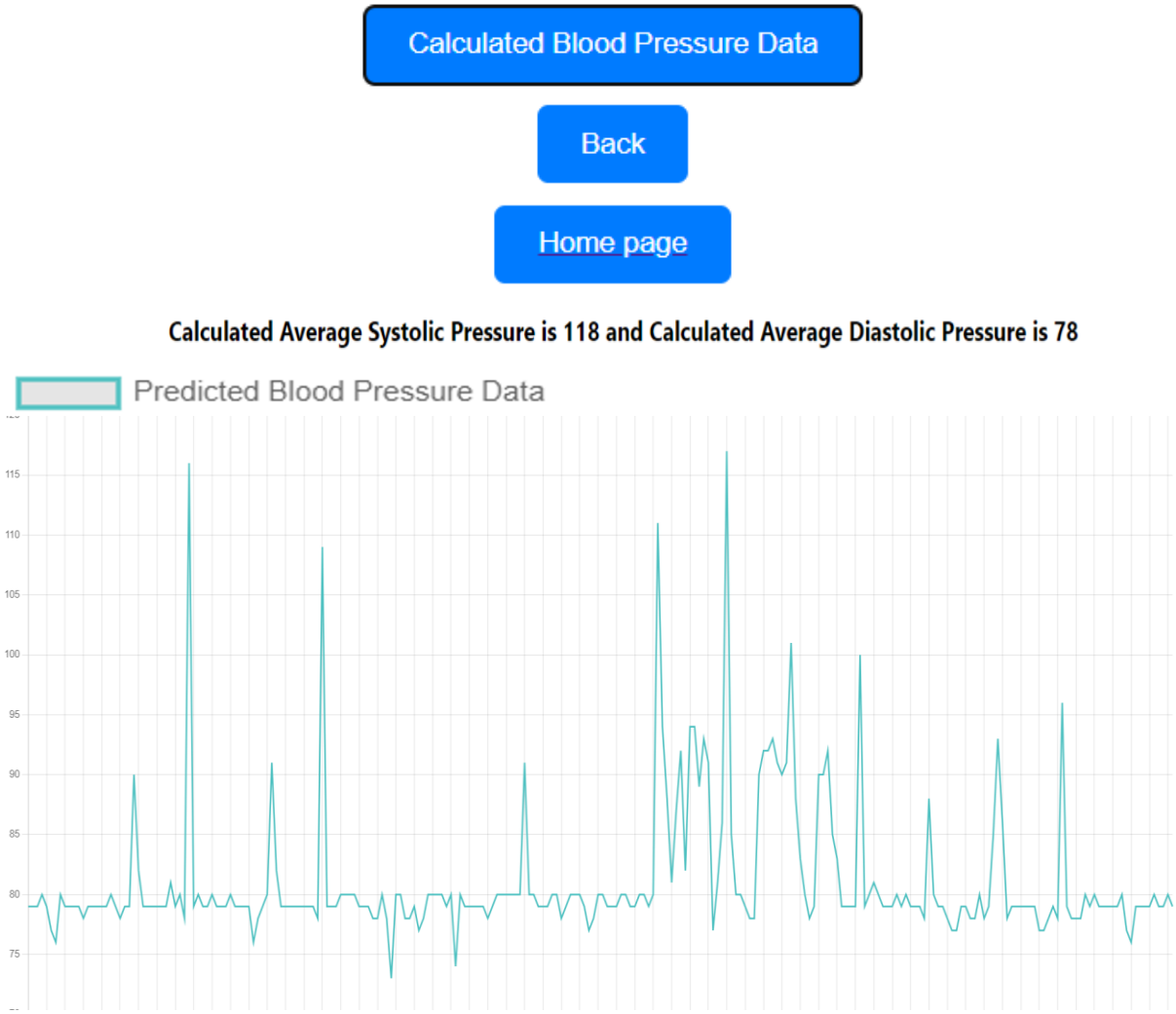


Figure 4.21: Estimated BP plot

id	ecg_value	ppg_value	predicted_bp
Filter	Filter	Filter	Filter
145...	2473	4095	75
145...	2512	4095	76
145...	2559	4095	76
145...	1990	4095	90
145...	1872	3373	81
145...	1831	2113	103
145...	1872	1326	89
145...	1879	1136	97
145...	1957	1291	96
145...	2010	1746	125
145...	2011	1762	124
145...	1890	2378	97
145...	1874	2654	85

Table 4.5: Estimated Blood Pressure for Different Data Points.

4.7 Evaluation:

To evaluate the developed methods further Method 1 and Method 3 were tested on 25 subjects to determine the Systolic and Diastolic BP, with their BP readings compared to those obtained from a Digital Sphygmomanometer.

Method	MAE of Systolic BP (mmHg)	MAE of Diastolic BP (mmHg)
Method 1	4.06	5.01
Method 3	6.62	4.32

* MAE – Mean Absolute Error

Method	RMSE of Systolic BP (mmHg)	RMSE of Diastolic BP (mmHg)
Method 1	4.69	5.89
Method 3	8.55	5.41

* RMSE – Root Mean Square Error

Table 4.6: Validation Results for Systolic and Diastolic Blood Pressure Estimation Methods on 30 Subjects.

Inference:

- Method 1 is better for Systolic BP with an MAE of 4.06 mmHg
- Method 3 is better for Diastolic BP with an MAE of 4.32 mmHg

In order to minimize the errors, the Systolic part of Method 1 and the Diastolic part of Method 3 were combined for optimizing the system. Thus, The Systolic BP estimation algorithm is developed from the Systolic estimation part of Method 1 and the Diastolic BP estimation algorithm is developed from the Diastolic estimation part of Method 3. This is used by the system for Systolic and Diastolic BP estimation.

Comparison Table:

Subject	Cuffless BP Monitor	Cuffed Digital BP Meter
1 F,19	105/76	109/65
	105/76	108/65
	107/76	109/70
2 M,29	128/77	130/85
	127/78	130/84
	126/77	129/85
3 M,41	121/74	118/75
	124/80	120/78
	123/80	119/78
4 M,35	123/77	117/70
	124/76	120/73
	124/76	120/72
5 M,38	116/74	117/75
	123/76	120/76
	120/75	118/75
6 M,55	124/78	129/87
	124/77	126/85
	123/78	126/85
7 M,21	120/78	121/78
	122/77	120/77
	121/76	121/78
8 F,45	119/77	110/75
	116/76	112/74
	112/78	110/75
9 F,52	115/76	111/71
	120/77	114/75
	123/77	116/78

10 F,30	125/75	128/77
	120/76	125/74
	127/76	126/77
11 F,40	124/76	128/85
	122/77	130/85
	119/76	128/81
12 M,28	118/76	116/77
	124/75	117/78
	123/76	117/78
13 M,27	123/78	129/82
	123/77	129/82
	125/77	130/80
14 M,31	120/78	115/75
	122/78	118/74
	123/78	118/75
15 M,50	126/77	130/90
	125/76	131/90
	127/77	130/90
16 F,28	122/73	110/70
	114/73	111/72
	115/76	112/72
17 F,35	121/74	120/71
	117/77	121/72
	123/75	121/72
18 M,33	123/76	125/80
	122/76	125/80
	121/75	124/80
19 M,36	128/74	122/74
	124/75	122/76
	123/76	121/75
20 F,51	124/78	130/82
	128/78	132/82
	123/77	130/81
21 M,46	121/73	115/77
	116/75	115/76
	120/81	116/77
22 M,45	116/77	119/78

	118/77	121/79
	118/77	116/79
23 F,43	105/76	106/70
	102/76	107/71
	110/78	109/70
24 M,21	125/74	124/78
	119/76	121/75
	120/73	123/74
25 F,63	129/76	140/89
	130/77	138/88
	130/77	140/87
26 M,37	123/74	118/70
	123/74	118/71
	125/75	119/70
27 F,43	121/79	123/76
	118/76	124/77
	123/75	125/78
28 M,24	117/78	113/74
	116/77	113/73
	116/77	113/74
29 F,63	109/77	102/70
	103/77	101/71
	105/76	101/70
30 M,40	126/79	130/83
	125/76	129/82
	123/77	129/82

Table 4.7: Comparison between Cuffless BP Monitor and Cuffed Digital BP Meter values

Arterial BP Estimation Algorithm	Mean Absolute Error (mmHg)
Systolic BP	4.06
Diastolic BP	4.32

Table 4.8: MAE of Arterial BP Estimation Algorithm

Arterial BP Estimation Algorithm	Mean Absolute Percentage Error (MAPE)
Systolic BP	3.37 %
Diastolic BP	5.53 %

Table 4.9: MAPE of Arterial BP Estimation Algorithm

The computed MAE for Systolic and Diastolic BP falls under the maximum standard allowable error mentioned by Association for Advancement of Medical Instrumentation i.e. MAE < 5 mmHg. This validation was confirmed by the findings of M. Jain, N. Kumar, and S. Deb in, “An affordable cuff-less blood pressure estimation solution,” [9]. The low MAPE values suggest that the algorithm is quite accurate in estimating both systolic and diastolic blood pressure values.

CHAPTER 5

CONCLUSION AND FUTURE WORKS

The developed system represents a significant advancement in remote physiological monitoring, leveraging wireless technology to seamlessly capture ECG and PPG signals and transmit them to a centralized server. By enabling patient-specific data storage and retrieval from anywhere, the system facilitates remote monitoring and enhances accessibility to critical health information.

The capability of the system is to estimate blood pressure demonstrates promising results, with estimations closely aligning with actual readings in most instances. While occasional discrepancies may arise in some cases, overall, the system provides reliable and valuable insights into the patient's cardiovascular health.

Furthermore, the user interface (UI) offers intuitive controls for managing device status and provides real-time monitoring of ECG, PPG, and blood pressure signal plots. By displaying numerical values of both systolic and diastolic blood pressure alongside graphical representations, the UI enhances user understanding and facilitates informed decision-making regarding patient care.

Overall, the developed system holds great potential in revolutionizing remote physiological monitoring, offering a user-friendly and reliable solution for healthcare professionals to monitor patients' cardiovascular health remotely. With further refinement and validation, this system could play a crucial role in improving healthcare delivery and patient outcomes.

APPENDIX A – CODE

ESP 32 Firmware for Data Acquisition

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

const char *ssid = "kamal";
const char *password = "kamal12345";
const char *server_ip = "192.168.1.5";
const int server_port = 80;
const char* server_address = "http://192.168.1.5/status";

const int ledPin = 2;

const int numSamples = 1000;
int esamples[numSamples];
int psamples[numSamples];

bool recordingStarted = false;

void setup() {
  pinMode(41, INPUT); // Setup for leads off detection LO +
  pinMode(40, INPUT); // Setup for leads off detection LO -
  pinMode(ledPin, OUTPUT);
  //pinMode(32, INPUT_PULLUP);
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}

void loop() {
  checkServerStatus();
  // Check server status to determine whether to start or stop recording
  if (recordingStarted) {
    // Start recording data
    recordData();
    delay(1000);
  }
}

void checkServerStatus() {
  // Send request to Flask server
  HTTPClient http;
  http.begin(server_address);
  int httpCode = http.GET();
  if (httpCode == HTTP_CODE_OK) {
    String payload = http.getString();
    Serial.println(payload);
```

APPENDIX A

```
    // Check if server flag is true
    if (payload.indexOf("true") != -1) {
      recordingStarted = true;
      digitalWrite(ledPin, HIGH);

      recordingStarted = false;
      digitalWrite(ledPin, LOW);
    }
  } else {
    Serial.printf("HTTP request failed, error: %s\n", http.errorToString(httpCode).c_str());
  }

  http.end();
}

void recordData() {
  int eval = 0;
  int pval = 0;

  // Collect samples
  for (int i = 0; i < numSamples; i++) {
    if ((digitalRead(40) == 1) || (digitalRead(41) == 1)) {
      eval = 0;
    } else {
      eval = analogRead(32);
    }
    pval = analogRead(36);
    delay(1);
    esamples[i] = eval;
    psamples[i] = pval;
    Serial.print(esamples[i]);      //the first variable for plotting
    Serial.print(",");              //seperator
    Serial.println(psamples[i]);          //the second variable for plotting including line break
    delay(20);
    }

    checkServerStatus();

    if(recordingStarted==false){
      HTTPClient http;


      http.begin("http://192.168.1.5/check_stop");
      int httpResponseCode = http.POST(""); // Send an empty body for POST
      if (httpResponseCode > 0) {
        String response = http.getString();
        Serial.println("HTTP Response code: " + String(httpResponseCode));
        Serial.println(response);
      } else {
        Serial.println("Error on HTTP request");
      }
      http.end();
        }


  // Send data to the server
  sendToServer(esamples, psamples, numSamples);
```

```
}

void sendToServer(int edata[], int pdata[], int dataSize) {
  WiFiClient client;
  if (client.connect(server_ip, server_port)) {
    DynamicJsonDocument jsonDoc(32768); // Increased document size to accommodate both
ECG and PPG data
    JsonArray ejsonArray = jsonDoc.createNestedArray("ecg_data");

    JsonArray pjsonArray = jsonDoc.createNestedArray("ppg_data");

    for (int i = 0; i < dataSize; i++) {
      ejsonArray.add(edata[i]);
      pjsonArray.add(pdata[i]);
    }

    String jsonPayload;
    serializeJson(jsonDoc, jsonPayload);
    String httpRequest = "POST /send_data HTTP/1.1\r\n";
    httpRequest += "Host: " + String(server_ip) + "\r\n";
    httpRequest += "Content-Type: application/json\r\n";
    httpRequest += "Content-Length: " + String(jsonPayload.length()) + "\r\n";
    httpRequest += "Connection: close\r\n\r\n";
    httpRequest += jsonPayload;

    client.print(httpRequest);

    Serial.println("Data sent to server");
    client.stop();
    Serial.println("Connection closed");
  } else {
    Serial.println("Failed to connect to server");
  }
}
```

Backend Server – Flask Framework

```
from flask import Flask, request, jsonify, render_template
import json
from flask_cors import CORS
import sqlite3
import random
import string
import pandas as pd
import tensorflow as tf
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from scipy.signal import find_peaks
from scipy.signal import argrelextrema

app = Flask(__name__)
start_flag = False
check_stop_flag = False
CORS(app, resources={r"/api/*": {"origins": "http://localhost:3000"}})
CORS(app, resources={r"/*": {"origins": "http://localhost:3000"}})

model = tf.keras.models.load_model("BPmodel.h5")
```

APPENDIX A

```
patient_id_s=''

def fetch_sensor_data(patient_id, column_name):
    try:
        # Connect to the SQLite database
        db_path = 'PPG_ECG_Data.db'  # Replace with the actual path
        connection = sqlite3.connect(db_path)

        cursor = connection.cursor()


        cursor.execute(f"SELECT {column_name} FROM sensor_values_{patient_id}

 ORDER    BY id DESC LIMIT 250;")

        rows = cursor.fetchall()

        cursor.close()
        connection.close()

        sensor_values = [row[0] for row in rows]
        return jsonify(sensor_values)

    except Exception as e:
        return jsonify({'error': str(e)}), 500

def fetch_average_values(patient_id):
    try:

        db_path = 'PPG_ECG_Data.db'
        connection = sqlite3.connect(db_path)

        cursor = connection.cursor()


        table_name = f'average_values_{patient_id}'

        cursor.execute(f"SELECT average_systolic, average_diastolic FROM {table_name}
ORDER BY id DESC LIMIT 1;")

        row = cursor.fetchone()
        cursor.close()
        connection.close()

        if row:
            average_values = {'average_systolic': row[0], 'average_diastolic': row[1]}
            return jsonify(average_values)
        else:
            return jsonify({'error': 'No data available'}), 404

    except Exception as e:
        return jsonify({'error': str(e)}), 500


def generate_patient_id():
    return 'PSG' + ''.join(random.choices(string.ascii_uppercase + string.digits, k=8))
```

APPENDIX A

```
def process_and_predict(ecg_data, ppg_data):
    timesteps = 1
    ecg_data = np.array(ecg_data).reshape(-1, 1)
    ppg_data = np.array(ppg_data).reshape(-1, 1)

    scaler = MinMaxScaler()
    merged_data = np.concatenate([ecg_data, ppg_data], axis=1)
    scaler.fit(merged_data)
    merged_data = scaler.transform(merged_data)
    merged_data = merged_data.reshape(-1, timesteps, merged_data.shape[1])

    predicted_bp = model.predict(merged_data)
    predicted_bp = predicted_bp.astype(int)
    predicted_bp = predicted_bp.reshape(-1)

    return predicted_bp

    def find_peaks_valleys(predicted_bp):
    highest_amplitude = max(predicted_bp)
    peaks, _ = find_peaks(predicted_bp, height=highest_amplitude - 10)
    valleys = argrelextrema(predicted_bp, np.less)

    systolic_values = predicted_bp[peaks]
    diastolic_values = predicted_bp[valleys]

    average_systolic = np.mean(systolic_values)
    average_diastolic = np.mean(diastolic_values)

    return average_systolic, average_diastolic

def patient_exists(patient_id):
    conn = sqlite3.connect('PPG_ECG_Data.db')
    cursor = conn.cursor()
    cursor.execute("SELECT COUNT(*) FROM patient_details WHERE patient_id = ?",
(patient_id,))
    result = cursor.fetchone()[0]
    conn.close()
    return result > 0

def create_table():
    conn = sqlite3.connect('PPG_ECG_Data.db')
    cursor = conn.cursor()
    cursor.execute('''
                CREATE TABLE IF NOT EXISTS patient_details (
                    patient_id TEXT PRIMARY KEY,
                    name TEXT,
                    age INTEGER,
                    gender TEXT
                )
            ''')
    conn.commit()
    conn.close()

create_table()
@app.route('/api/new_patient', methods=['POST'])
def add_patient_details():
    global patient_id_s
```

APPENDIX A

```
    if request.method == 'POST':
        try:
            # Get JSON data from the request body
            data = request.get_json()
            print("Patient Details Received ")

            if data is not None and all(key in data for key in ['age', 'gender', 'name']):
                age = data['age']
                gender = data['gender']
                name = data['name']
                patient_id = generate_patient_id()

                # Connect to the database
                conn = sqlite3.connect('PPG_ECG_Data.db')
                cursor = conn.cursor()

                # Insert the patient details into the database
                cursor.execute('INSERT INTO patient_details (patient_id, name, age, gender)
VALUES (?, ?, ?, ?)',
                               (patient_id, name, age, gender))

                # Create sensor_values table for the patient
                cursor.execute(f"CREATE TABLE IF NOT EXISTS sensor_values_{patient_id} (id
INTEGER PRIMARY KEY AUTOINCREMENT, ecg_value INTEGER, ppg_value INTEGER,
predicted_bp INTEGER)")

                # Create average_values table for the patient
                cursor.execute(f"CREATE TABLE IF NOT EXISTS average_values_{patient_id}
(id INTEGER PRIMARY KEY AUTOINCREMENT, average_diastolic INTEGER,
average_systolic INTEGER)")

                conn.commit()
                conn.close()
                patient_id_s=patient_id
                return jsonify({'patient_id': patient_id}), 200
            else:
                return 'Invalid JSON format. Missing age, gender, or name.', 400
        except Exception as e:
            return f'Error adding patient details: {str(e)}', 400
    else:
        return 'Method not allowed', 405


@app.route('/api/edit_patient', methods=['PUT'])
def edit_patient_details():
    global patient_id_s
    if request.method == 'PUT':
        try:
            # Get JSON data from the request body
            data = request.get_json()
            print("Patient Details Received ")

            if data is not None and all(key in data for key in ['patient id', 'age', 'gender', 'name']):
                age = data['age']
                gender = data['gender']
                name = data['name']
```

APPENDIX A

```
            patient_id = data['patient id']
            patient_id_s=patient_id
            conn = sqlite3.connect('PPG_ECG_Data.db')
            cursor = conn.cursor()

            # Check if the patient ID exists in the database
            cursor.execute('SELECT * FROM patient_details WHERE patient_id = ?',
(patient_id,))
            existing_patient = cursor.fetchone()
            if existing_patient:
                # Update patient details
                cursor.execute('UPDATE patient_details SET name = ?, age = ?, gender = ?
WHERE patient_id = ?',
                               (name, age, gender, patient_id))
                conn.commit()
                conn.close()
                return jsonify({'message': 'Patient details updated successfully'}), 200
            else:
                conn.close()
                return jsonify({'error': 'Patient does not exist'}), 404
        else:
            return 'Invalid JSON format. Missing a Required Field', 400
    except Exception as e:
        return f'Error editing patient details: {str(e)}', 400

else:
    return 'Method not allowed', 405

@app.route('/verify_patient', methods=['POST'])
def verify_patient():
    if request.method == 'POST':
        try:
            data = request.get_json()
            if 'patient_id' in data:
                patient_id = data['patient_id']
                if patient_exists(patient_id):
                    return jsonify({'exists': True}), 200
                else:
                    return jsonify({'exists': False}), 200
            else:
                return 'Invalid JSON format. Missing patient_id.', 400
        except Exception as e:
            return f'Error verifying patient: {str(e)}', 400
    else:
        return 'Method not allowed', 405

@app.route('/send_data', methods=['POST'])
def receive_data():
    global patient_id_s
    patient_id=patient_id_s
    print(patient_id)
    if request.method == 'POST':
        try:
            # Get JSON data from the request body
            data = request.get_json()
            print("Data Received ")
```

APPENDIX A

```
            if data is not None and 'ecg_data' in data and 'ppg_data' in data:
                ecg_values = data['ecg_data']
                ppg_values = data['ppg_data']
                predicted_bp_group = process_and_predict(ecg_values, ppg_values)

                conn = sqlite3.connect('PPG_ECG_Data.db')
                cursor = conn.cursor()
                print("BP Prediction Done")

                # Create a new sensor_values table for the patient if it doesn't exist
                cursor.execute(f"CREATE TABLE IF NOT EXISTS sensor_values_{patient_id} (id
INTEGER PRIMARY KEY AUTOINCREMENT, ecg_value INTEGER, ppg_value INTEGER,
predicted_bp INTEGER)")

                print("Table Created")

                # Insert data into the patient-specific table
                for ecg_value, ppg_value, pred_bp in zip(ecg_values, ppg_values,
predicted_bp_group):
                    print(f"Inserting data: ecg_value={ecg_value}, ppg_value={ppg_value},
pred_bp={pred_bp}")
                    cursor.execute(f'INSERT INTO sensor_values_{patient_id} (ecg_value,
ppg_value, predicted_bp) VALUES (?, ?, ?)', (ecg_value, ppg_value, int(pred_bp)))
                    print("Data inserted successfully!")

                conn.commit()

                average_systolic, average_diastolic = find_peaks_valleys(predicted_bp_group)

                print(f"Average Systolic BP : {average_systolic}")
                print(f"Average Diastolic BP : {average_diastolic}")
                cursor.execute(f'INSERT INTO average_values_{patient_id} (average_diastolic,
average_systolic) VALUES (?, ?)', (int(average_diastolic), int(average_systolic)))


                conn.commit()
                conn.close()
                return 'Data received and processed successfully'
            else:
                return 'Invalid JSON format. Missing patient_id, ecg_data, or ppg_data.', 400
        except Exception as e:
            return f'Error processing data: {str(e)}', 400
    else:
        return 'Method not allowed', 405

@app.route('/patient_id_api/<patient_id>', methods=['POST'])
def set_patient_id(patient_id):
    global patient_id_s
    patient_id_s = patient_id
    print("Patient ID received:", patient_id_s)
    return jsonify({'message': 'Patient ID set successfully'}), 200

@app.route('/measure')
def measure():
    global start_flag
    global check_stop_flag
    check_stop_flag = False
```

APPENDIX A

```
    start_flag = True
    return jsonify({'status': 'success', 'message': 'measure started'})

@app.route('/status')
def status():
    return jsonify({'start_flag': start_flag})

@app.route('/stop')
def stop():
    global start_flag
    start_flag = False
    return jsonify({'status': 'success', 'message': 'measure stopped'})

@app.route('/check_stop', methods=['POST'])
def checkstop():
    global check_stop_flag
    check_stop_flag = True
    return jsonify({'check_stop_flag': check_stop_flag})
    #return jsonify({'status': 'success', 'message': 'measure started'})


@app.route('/status_stop')
def statusstop():
    global check_stop_flag
    return jsonify({'check_stop_flag': check_stop_flag})


@app.route('/api/average_bp/<patient_id>', methods=['GET'])
def get_latest_average_values(patient_id):
    try:
        response = fetch_average_values(patient_id)
        return response

    except Exception as e:
        print(f"Error in get_latest_average_values: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/ppg/<patient_id>', methods=['GET'])
def get_ppg_data(patient_id):
    try:
        response = fetch_sensor_data(patient_id, 'ppg_value')
        return response
    except Exception as e:
        print(f"Error in get_ppg_data: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/ecg/<patient_id>', methods=['GET'])
def get_ecg_data(patient_id):
    try:
        response = fetch_sensor_data(patient_id, 'ecg_value')
        return response
    except Exception as e:
        print(f"Error in get_ecg_data: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/calculated_bp/<patient_id>', methods=['GET'])
def get_calculated_bp_data(patient_id):
```

```
    try:
        response = fetch_sensor_data(patient_id, 'predicted_bp')
        return response
    except Exception as e:
        print(f"Error in get_calculated_bp_data: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80, debug=True)
```

Frontend – React Framework

App.js

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from './Home';
import PatientRegistration from './PatientRegistration';
import MeasurementComponent from './MeasurementComponent';
import SelectPatient from './SelectPatient';
import ECGData from './ECGData';
import PPGData from './PPGData';
import BPData from './BPData';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/register" element={<PatientRegistration />} />
        <Route path="/select-patient" element={<SelectPatient />} />
        <Route path="/measurement/:patientId" element={<MeasurementComponent />} />
        <Route path="/ecg-data/:patientId" element={<ECGData />} />
        <Route path="/ppg-data/:patientId" element={<PPGData />} />
        <Route path="/bp-data/:patientId" element={<BPData />} />
        <Route path="/" element={<Home />} />
      </Routes>
    </Router>
  );
}

export default App;
```

PatientRegistration.js

```
import React, { useState } from 'react';
import axios from 'axios';
import './PatientRegistration.css'; // Import CSS file for styling

function PatientRegistration() {
  const [formData, setFormData] = useState({
    name: '',
    age: '',
    gender: ''
```

```
  });
  const [message, setMessage] = useState('');

  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value
    });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('http://127.0.0.1:80/api/new_patient', formData);
      setMessage(`Patient ID: ${response.data.patient_id}. Patient details added
successfully.`);
    } catch (error) {
      setMessage(`Error adding patient details: ${error.response.data}`);
    }
  };

  return (
    <div className="registration-container">
      <h2>Patient Registration</h2>
      <form onSubmit={handleSubmit} className="registration-form">
        <label className="form-label">
        <label className="form-label">
          Age   :
          <input type="number" name="age" value={formData.age} onChange={handleChange}
required />
        </label>
        <br />
        <label className="form-label">
          Gender :
          <select name="gender" value={formData.gender} onChange={handleChange}
required>
            <option value="">Select Gender</option>
            <option value="male">Male</option>
            <option value="female">Female</option>
            <option value="other">Other</option>

          </select>
        </label>
        <br />
        <div className="button-container">
          <button type="submit">Register</button>
          <button type="button" onClick={() => window.history.back()}>Back</button> {/* Back
button */}
        </div>
      </form>
  );
}

export default PatientRegistration;
```

MeasureComponent.js

APPENDIX A

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useParams } from 'react-router-dom';
import './MeasurementComponent.css'; // Import CSS file

const MeasurementComponent = () => {
  const [message, setMessage] = useState('');
  const [startFlag, setStartFlag] = useState(false);
  const { patientId } = useParams();

  useEffect(() => {
    // Fetch patient data or perform other operations based on patientId
  }, [patientId]);

  const handleMeasureStart = async () => {
    try {

        setMessage("Measuring Starting Do not Remove Your finger!");
        await new Promise(resolve => setTimeout(resolve, 4000));
        await axios.get('http://127.0.0.1:80/measure');
        await axios.post(`http://127.0.0.1:80/patient_id_api/${patientId}`);
        setStartFlag(true);
        setMessage('Measuring started');
    } catch (error) {
        console.error('Error starting measure:', error);
        setMessage(`Error starting measure: ${error.message}`);
    }
};

  const handleMeasureStop = async () => {
    try {

      setMessage("Measuring Stopping. Do not remove your finger!");
      await new Promise(resolve => setTimeout(resolve, 3000));
      await axios.get('http://127.0.0.1:80/stop');
      // Define a function to check the status repeatedly until conditions are satisfied
      const checkStatus = async () => {
        try {
          // Call the API endpoint to check status
          const statusResponse = await axios.get('http://127.0.0.1:80/status_stop');
          const { check_stop_flag } = statusResponse.data;
          // If check_stop_flag is true, update message and set start flag to false
          if (check_stop_flag) {
            setMessage('Measuring Stopped');
            setStartFlag(false);
          } else {
            // If check_stop_flag is false, wait for 1 second and check again
            setTimeout(checkStatus, 1000); // Wait for 1 second before checking again
          }
        } catch (error) {
          console.error('Error checking measure status:', error);
          setMessage(`Error checking measure status: ${error.message}`);
        }
      };
      // Start checking status
      await checkStatus();
    } catch (error) {
```

```
      console.error('Error stopping measure:', error);
      setMessage(`Error stopping measure: ${error.message}`);
    }
  };

  const checkStatus = async () => {
    try {
      // Call the API endpoint to check status
      const response = await axios.get('http://127.0.0.1:80/status');
      setStartFlag(response.data.start_flag);
    } catch (error) {
      console.error('Error checking status:', error);
      setMessage(`Error checking status: ${error.message}`);
    }
  };

  const navigateTo = (path) => {
    window.location.href = path;
  };

  return (
    <div className="measurement-container">
      <div className="button-measurement-container">
        <button onClick={() => navigateTo(`/ecg-data/${patientId}`)}>ECG</button>
        <button onClick={() => navigateTo(`/ppg-data/${patientId}`)}>PPG</button>
        <button onClick={() => navigateTo(`/bp-data/${patientId}`)}>BP</button>
        <button onClick={handleMeasureStart}>Start Measure</button>
        <button onClick={handleMeasureStop}>Stop Measure</button>
      </div>

      {/* Additional rendering based on conditions */}
      {message && <p className="message">{message}</p>}

      {/* Button for navigating to home page */}
      <button onClick={() => navigateTo("/")} className="home-link">Home page</button>
    </div>
  );
};

export default MeasurementComponent;
```

SelectPatient.js

```
// SelectPatient.js

import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import MeasurementComponent from './MeasurementComponent';

import './SelectPatient.css'; // Import the CSS file

function SelectPatient() {
  const [patientId, setPatientId] = useState('');
  const [verificationResult, setVerificationResult] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (event) => {
```

```
    event.preventDefault();
    try {
      const response = await fetch('http://127.0.0.1:80/verify_patient', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ patient_id: patientId }),
      });

      if (response.ok) {
        const data = await response.json();
        if (data.exists) {
          navigate(`/measurement/${patientId}`);
        } else {
          setVerificationResult('Patient does not exist');
        }
      } else {
        throw new Error('Failed to verify patient');
      }
    } catch (error) {
      console.error('Error verifying patient:', error);
      setVerificationResult('Error verifying patient. Please try again.');
    }
  };

  const handleChange = (event) => {
    setPatientId(event.target.value);
    setVerificationResult('');
  };

  return (
    <div className="ap-container">
      <h2>Select Patient</h2>
      <form onSubmit={handleSubmit} className="form-container">
        <label htmlFor="patientId" className="label">Enter Patient ID:</label>
        <button type="submit" className="button">Submit</button>
      </form>

      {verificationResult && <p className="verification">{verificationResult}</p>}
    </div>
  );
}

export default SelectPatient;
```

ECGData.js

```
import React, { useState, useEffect, useRef } from 'react';
import axios from 'axios';
import Chart from 'chart.js/auto';
import { useParams, Link, useNavigate } from 'react-router-dom';
import './PatientRegistration.css';
import './ButtonContainer.css';

function ECGData() {
  const { patientId } = useParams();
```

APPENDIX A

```
const [data, setData] = useState([]);
const [message, setMessage] = useState('');
const [chartLabel, setChartLabel] = useState('');
const [renderChartFlag, setRenderChartFlag] = useState(false);
const [startFlag, setStartFlag] = useState(false);
const [intervalId, setIntervalId] = useState(null);

useEffect(() => {
  return () => clearInterval(intervalId);
}, [intervalId]);

useEffect(() => {
  if (data.length > 0 && chartRef.current && renderChartFlag) {
    renderChart();
  }
}, [data, renderChartFlag]);

const fetchData = async (endpoint) => {
  if (!patientId) return;
  try {
    const response = await axios.get(`http://127.0.0.1:80/api/${endpoint}/${patientId}`);
    setData(response.data);
    setMessage('');
    setRenderChartFlag(true);
  } catch (error) {
    console.error('Error fetching data:', error);
    setMessage(`Error fetching data: ${error.message}`);
  }
};

const navigate = useNavigate();

const handleButtonClick = () => {
  fetchData('ecg');
  setChartLabel('ECG Data');
  const id = setInterval(() => {
    fetchData('ecg');
  }, 5000);
  setIntervalId(id);

};

const renderChart = () => {
  const ctx = chartRef.current.getContext('2d');

  if (chartInstance.current) {
    chartInstance.current.destroy();
  }

  const chartData = {
    labels: data.map((_, index) => index + 1),
    datasets: [
      {
        label: chartLabel,
        data: data,
        borderColor: 'rgba(75, 192, 192, 1)',
```

```
        borderWidth: 2,
        fill: false,
        pointStyle: 'hidden'
      },
    ],
  };

  chartInstance.current = new Chart(ctx, {
    type: 'line',
    data: chartData,
    options: {
      elements: {
        point: {
          radius: 0
        }
      }
    }
  });
};

 const handleBack = () => {
  setData([]);
  setMessage('');
  setChartLabel('');
  if (chartInstance.current) {
    chartInstance.current.destroy();
  }
  setRenderChartFlag(false);
  clearInterval(intervalId);
  navigate(-1);
};

 return (
  <div className="app-container">
    <div className="button-container">
      <button onClick={handleButtonClick}>Plot ECG Data</button>
      <button onClick={handleBack}>Back</button>
      <Link to="/">
    </div>
  {message && <p style={{ fontWeight: 'bold' }}>{message}</p>}
  {renderChartFlag && <canvas id="myChart" width="400" height="200"
ref={chartRef}></canvas>}

  </div>
 );

export default ECGData;
```

PPGData.js

```
import React, { useState, useEffect, useRef } from 'react';
import axios from 'axios';
import Chart from 'chart.js/auto';
import { useParams, Link, useNavigate } from 'react-router-dom';
import './PatientRegistration.css';
import './ButtonContainer.css';
```

APPENDIX A

```
function PPGData() {
  const { patientId } = useParams();
  const [data, setData] = useState([]);
  const [message, setMessage] = useState('');
  const [chartLabel, setChartLabel] = useState('');
  const [renderChartFlag, setRenderChartFlag] = useState(false);
  const chartRef = useRef(null);
  const chartInstance = useRef(null);
  const [startFlag, setStartFlag] = useState(false);
  const [intervalId, setIntervalId] = useState(null);

  useEffect(() => {
    return () => clearInterval(intervalId);
  }, [intervalId]);

  useEffect(() => {
    if (data.length > 0 && chartRef.current && renderChartFlag) {
      renderChart();
    }
  }, [data, renderChartFlag]);

  const fetchData = async (endpoint) => {
    if (!patientId) return;
    try {
      const response = await axios.get(`http://127.0.0.1:80/api/${endpoint}/${patientId}`);
      setData(response.data);
      setMessage('');
      setRenderChartFlag(true);
    } catch (error) {
      console.error('Error fetching data:', error);
      setMessage(`Error fetching data: ${error.message}`);
    }
  };

  const navigate = useNavigate();

  const handleButtonClick = () => {
    fetchData('ppg');
    setChartLabel('PPG Data');
    const id = setInterval(() => {
      fetchData('ppg');
    }, 5000);
    setIntervalId(id);

  };

  const renderChart = () => {
    const ctx = chartRef.current.getContext('2d');

    if (chartInstance.current) {
      chartInstance.current.destroy();
    }

    const chartData = {
      labels: data.map((_, index) => index + 1),
      datasets: [
        {
```

```
          label: chartLabel,
          data: data,
          borderColor: 'rgba(75, 192, 192, 1)',
          borderWidth: 2,
          fill: false,
          pointStyle: 'hidden'

        },
      ],
    };

    chartInstance.current = new Chart(ctx, {
      type: 'line',
      data: chartData,
      options: {
        elements: {
          point: {
            radius: 0
          }
        }
      }
    });
  };

  const handleBack = () => {
    setData([]);
    setMessage('');
    setChartLabel('');
    if (chartInstance.current) {
      chartInstance.current.destroy();
    }
    setRenderChartFlag(false);
    clearInterval(intervalId);
    navigate(-1);
  };

  return (
    <div className="app-container">
      <div className="button-container">
        <button className="button-container" onClick={handleButtonClick}>Plot PPG
Data</button>
        <button className="button-container" onClick={handleBack}>Back</button>
        <Link to="/">
          <button className="button-container">Home page</button>
        </Link>
      </div>

      {message && <p style={{ fontWeight: 'bold' }}>{message}</p>}

      {renderChartFlag && <canvas id="myChart" width="400" height="200"
ref={chartRef}></canvas>}

    </div>
  );
}
```

```
export default PPGData;
```

BPData.js

```
import React, { useState, useEffect, useRef } from 'react';
import axios from 'axios';
import Chart from 'chart.js/auto';
import { useParams, Link, useNavigate } from 'react-router-dom';
import './PatientRegistration.css';
import './ButtonContainer.css';

function BPData() {
  const { patientId } = useParams();
  const [data, setData] = useState([]);
  const [message, setMessage] = useState('');
  const [chartLabel, setChartLabel] = useState('');
  const [renderChartFlag, setRenderChartFlag] = useState(false);
  const chartRef = useRef(null);
  const chartInstance = useRef(null);
  const [startFlag, setStartFlag] = useState(false);
  const [intervalId, setIntervalId] = useState(null);

  useEffect(() => {
    return () => clearInterval(intervalId);
  }, [intervalId]);

  useEffect(() => {
    if (data.length > 0 && chartRef.current && renderChartFlag) {
      renderChart();
    }
  }, [data, renderChartFlag]);

  const fetchData = async (endpoint) => {
    if (!patientId) return;
    try {
      const response = await axios.get(`http://127.0.0.1:80/api/${endpoint}/${patientId}`);
      setData(response.data);
      setMessage('');
      setRenderChartFlag(true);
    } catch (error) {
      console.error('Error fetching data:', error);
      setMessage(`Error fetching data: ${error.message}`);
    }
  };

  const fetchAverageAndPredictedBPData = async () => {
    try {
      const averageBPResponse = await
axios.get(`http://127.0.0.1:80/api/average_bp/${patientId}`);
      const { average_systolic, average_diastolic } = averageBPResponse.data;

      const predictedBPResponse = await
axios.get(`http://127.0.0.1:80/api/calculated_bp/${patientId}`);

      setMessage(`Calculated Average Systolic Pressure is ${average_systolic} and
Calculated Average Diastolic Pressure is ${average_diastolic}`);
      setData(predictedBPResponse.data);
```

```
      setChartLabel('Predicted Blood Pressure Data');
      setRenderChartFlag(true);
    } catch (error) {
      console.error('Error fetching average and predicted BP data:', error);
      setMessage(`Error fetching average and predicted BP data: ${error.message}`);
    }
  };

  const handleButtonClick = () => {
    fetchAverageAndPredictedBPData();
    const id = setInterval(fetchAverageAndPredictedBPData, 5000);
    setIntervalId(id);
  };

  const renderChart = () => {
    const ctx = chartRef.current.getContext('2d');

    if (chartInstance.current) {
      chartInstance.current.destroy();
    }

    const chartData = {
      labels: data.map((_, index) => index + 1),
      datasets: [
        {
          label: chartLabel,
          data: data,
          borderColor: 'rgba(75, 192, 192, 1)',
          borderWidth: 2,
          fill: false,
          pointStyle: 'hidden'
      ],
    };

    chartInstance.current = new Chart(ctx, {
      type: 'line',
      data: chartData,

      }
    });
  };

  const navigate = useNavigate();

  const handleBack = () => {
    setData([]);
    setMessage('');
    setChartLabel('');
    if (chartInstance.current) {
      chartInstance.current.destroy();
    }
    setRenderChartFlag(false);
    clearInterval(intervalId);
    navigate(-1);
  };

  return (
```

APPENDIX A

```
    <div className="app-container">
      <div className="button-container">
        <button onClick={handleButtonClick}>Calculated Blood Pressure Data</button>
        <button onClick={handleBack}>Back</button>
        <Link to="/">
          <button className="button-container">Home page</button>
        </Link>
      </div>

      {message && <p style={{ fontWeight: 'bold' }}>{message}</p>}

      {renderChartFlag && <canvas id="myChart" width="400" height="200"
ref={chartRef}></canvas>}


    </div>
  );
}

export default BPData;
```

APPENDIX B – DATASHEETS

AD8232 DATASHEET

ANALOG DEVICES

Single-Lead, Heart Rate Monitor Front End

Data Sheet **AD8232**

FEATURES

- Fully integrated single-lead ECG front end
- Low supply current: 170 µA (typical)
- Common-mode rejection ratio: 80 dB (dc to 60 Hz)
- Two or three electrode configurations
- High signal gain (G = 100) with dc blocking capabilities
- 2-pole adjustable high-pass filter
- Accepts up to ±300 mV of half cell potential
- Fast restore feature improves filter settling
- Uncommitted op amp
- 3-pole adjustable low-pass filter with adjustable gain
- Leads off detection: ac or dc options
- Integrated right leg drive (RLD) amplifier
- Single-supply operation: 2.0 V to 3.5 V
- Integrated reference buffer generates virtual ground
- Rail-to-rail output
- Internal RFI filter
- 8 kV HBM ESD rating
- Shutdown pin
- 20-lead, 4 mm × 4 mm LFCSP and LFCSP_SS package
- Qualified for automotive applications

APPLICATIONS

- Fitness and activity heart rate monitors
- Portable ECG
- Remote health monitors
- Gaming peripherals
- Biopotential signal acquisition

FUNCTIONAL BLOCK DIAGRAM

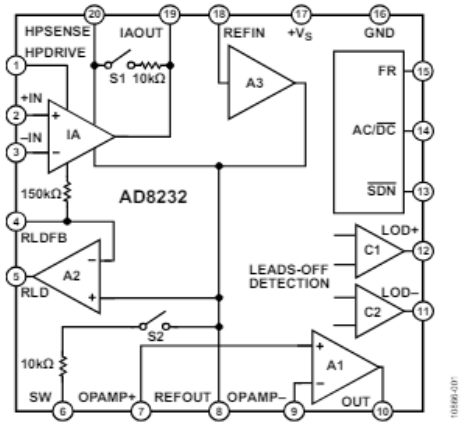


Figure 1.

GENERAL DESCRIPTION

The AD8232 is an integrated signal conditioning block for ECG and other biopotential measurement applications. It is designed to extract, amplify, and filter small biopotential signals in the presence of noisy conditions, such as those created by motion or remote electrode placement. This design allows for an ultralow power analog-to-digital converter (ADC) or an embedded microcontroller to acquire the output signal easily.

The AD8232 can implement a two-pole high-pass filter for eliminating motion artifacts and the electrode half-cell potential. This filter is tightly coupled with the instrumentation architecture of the amplifier to allow both large gain and high-pass filtering in a single stage, thereby saving space and cost.

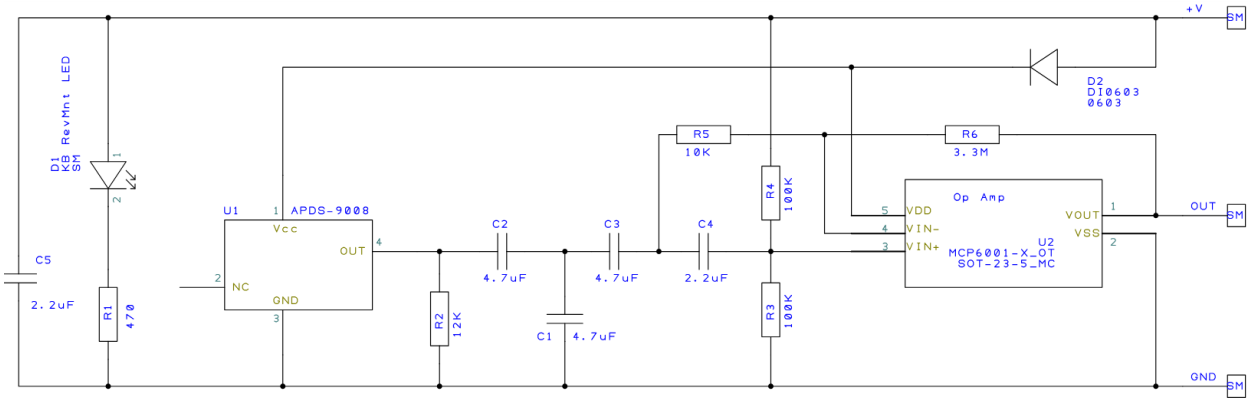
An uncommitted operational amplifier enables the AD8232 to create a three-pole low-pass filter to remove additional noise. The user can select the frequency cutoff of all filters to suit different types of applications.

To improve common-mode rejection of the line frequencies in the system and other undesired interferences, the AD8232 includes an amplifier for driven lead applications, such as right leg drive (RLD).

The AD8232 includes a fast restore function that reduces the duration of otherwise long settling tails of the high-pass filters. After an abrupt signal change that rails the amplifier (such as a leads off condition), the AD8232 automatically adjusts to a higher filter cutoff. This feature allows the AD8232 to recover quickly, and therefore, to take valid measurements soon after connecting the electrodes to the subject.

The AD8232 is available in a 4 mm × 4 mm, 20-lead LFCSP and a LFCSP_SS package. Performance for the A grade models is specified from 0°C to 70°C and the models are operational from −40°C to +85°C. Performance for the W grade models is specified over the automotive temperature range of −40°C to +105°C.

SEN11574 DATASHEET



WORLD FAMOUS ELECTRONICS llc.

www.pulsesensor.com

PULSE SENSOR
EASY TO USE HEART RATE SENSOR & KIT

General Description	Features
The Pulse Sensor is the original low-cost optical heart rate sensor (PPG) for Arduino and other microcontrollers. It's designed and made by World Famous Electronics, who actively maintain extensive example projects and code at: www.pulsesensor.com	• Includes Kit accessories for high-quality sensor readings • Designed for Plug and Play • Small size and embeddable into wearables • Works with any MCU with an ADC • Works with 3 Volts or 5 Volts • Well-documented Arduino library

Absolute Maximum Ratings	Min	Typ	Max	Unit
Operating Temperature Range	-40		+85	°C
Input Voltage Range	3		5.5	V
Output Voltage Range	0.3	Vdd/2	Vdd	V
Supply Current	3		4	mA

Family Name	**Sensor Modules**
Width	16 mm
Length	600 mm , Height 3.1 mm
Kit Contents	Pulse Sensor Board, 600 mm Cable, Ear Clip, Hook and Loop Finger Strap, Sensor Protection Stickers

BIBLIOGRAPHY

[1] Erem C., Hacihasanoglu A., Kocak M., Deger O., Topbas M. Prevalence of prehypertension and hypertension and associated risk factors among Turkish adults: trabzon hypertension study. Journal of Public Health. 2009;31(1):47–58. doi: 10.1093/pubmed/fdn078.

[2] ISO 81060-1:2007. Non-invasive sphygmomanometers-Part 1: Requirements and test methods for non-automated measurement type. Int Organization Standardization 2007.

[3] Myers MG. Automated blood pressure measurement in routine clinical practice. Blood Press Monit. 2006; 11:59–62.

[4] Personalized anomaly detection in PPG data using representation learning and biometric identification March 2024Biomedical Signal Processing and Control 94:10621.

[5] Scarborough WR. Proposals for Ballistocardiographic Nomenclature and Conventions: Revised and Extended: Report of Committee on Ballistocardiographic Terminology. Circulation. 1956; 14:435–450.

[6] D. T. Phan et al., “Non-invasive, wearable multi biosensors for continuous, long-term monitoring of blood pressure via internet of things applications,” Computers and Electrical Engineering, vol. 102, p. 108187, 2022. doi: 10.1016/j.compeleceng.2022.108187

[7] A. Jain, M. Singh, and B. Singh, “Real time system on chip based wearable cardiac activity monitoring sensor,” Measurement: Sensors, vol.24, p. 100523, 2022. doi: 10.1016/j.measen.2022.100523

[8] P. Bellier et al., “A wireless low-power single-unit wearable system for continuous early warning score calculation,” IEEE Sensors Journal, vol. 23, no. 11, pp. 12171–12180, 2023. doi:10.1109/jsen.2023.3267146

[9] C. Qiu, T. Wu, F. Heydari, J.-M. Redoute, and M. R. Yuce, “Wearable blood pressure monitoring based on bio-impedance and photoplethysmography sensors on the arm,” 2018 IEEE SENSORS, 2018. doi:10.1109/icsens.2018.8589785

[10] S. Critcher and T. J. Freeborn, “Localized bioimpedance measurements with the MAX3000x integrated circuit: Characterization and demonstration,” Sensors, vol. 21, no. 9, p. 3013, 2021. doi:10.3390/s21093013

BIBLIOGRAPHY

[11] K. Lee et al., “Neckband-based continuous blood pressure monitoring device with offset-tolerant ROIC,” IEEE Access, vol. 10, pp. 17300–17309, 2022. doi:10.1109/access.2022.3149925

[12] B. Ibrahim and R. Jafari, “Cuffless blood pressure monitoring from an array of wrist bio-impedance sensors using subject-specific regression models: Proof of concept,” IEEE Transactions on Biomedical Circuits and Systems, vol. 13, no. 6, pp. 1723–1735, 2019. doi:10.1109/tbcas.2019.2946661

[13] D. Osman, M. Jankovic, K. Sel, R. I. Pettigrew, and R. Jafari, “Blood pressure estimation using a single channel Bio-Impedance Ring Sensor,” 2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), 2022. doi:10.1109/embc48229.2022.9871653

[14] Abebe S. M., Berhane Y., Worku A., Getachew A. Prevalence and associated factors of hypertension: a crossectional community based study in Northwest Ethiopia. PLoS ONE. 2015;10(4) doi: 10.1371/journal.pone. 0125210.e0125210

[15] Mendis S. World Health Organisation; 2010. Global status report on non-communicable diseases 2010. http://www.who.int/nmh/publications/ncd_report2010/en.

[16] Tabrizi J. S., Sadeghi-Bazargani H., Farahbakhsh M., Nikniaz L., Nikniaz Z. Prevalence and associated factors of prehypertension and hypertension in Iranian population: the lifestyle promotion project (LPP) PLoS ONE. 2016;11(10) doi: 10.1371/journal.pone. 0165264.e0165264.

[17] Kumar M. R., Shankar R., Singh S. Hypertension among the adults in rural Varanasi: a cross-sectional study on prevalence and health seeking behavior. Indian Journal of Preventive and Social Medicine. 2016;47(1-2):78–83.

[18] Chobanian A. V., Bakris G. L., Black H. R., et al. Seventh report of the Joint National Committee on prevention, detection, evaluation, and treatment of high blood pressure. Hypertension. 2003;42(6):1206–1252. doi: 10.1161/01.HYP.0000107251. 49515.c2.

[19] Prabakaran J., Vijayalakshmi N., VenkataRao E. Prevalence of hypertension among urban adult population (25–64 years) of Nellore. International Journal of Research & Development of Health. 2013;1(2):42–49.

BIBLIOGRAPHY

[20]. Fisher N. D., Williams G. H. Hypertensive vascular disease. In: Kasper D. L., Braunwald E., Fauci A. S., et al., editors. Harrison's Principles of Internal Medicine. 16th. New York, NY, USA: McGraw-Hill; 2005. pp. 1463–1481.

PLAGIARISM REPORT

turnitin

Similarity Report ID: oid:22828:58314813

PAPER NAME	AUTHOR
REPORT BATCH_12_.docx	**Ashwin B**

WORD COUNT	CHARACTER COUNT
10071 Words	**57983 Characters**

PAGE COUNT	FILE SIZE
65 Pages	**7.2MB**

SUBMISSION DATE	REPORT DATE
Apr 27, 2024 11:50 AM GMT+5:30	**Apr 27, 2024 11:51 AM GMT+5:30**

18% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 9% Internet database
- 8% Publications database
- Crossref database
- Crossref Posted Content database
- 15% Submitted Works database