We hereby declare that this work has not been submitted for any other degree/course at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original and entirely the result of my own work at the University of Strathclyde under the supervision of Dr David McMillan."

GP19520: Wireless Cardiac Auscultation and At-Home Monitoring System

Megan Toney – 202007764
Isla Macdonald - 202009342
Zainab Munir – 201937380
Jakub Manda –202009261
Michail Kasmeridis – 201902441

Supervisor: Dr David McMillan

Date: April 2025

Abstract

CONTENTS

Table of Figures (Figures)

Table of Figures (Tables)

Table of Acronyms

ACRONYM	MEANING
ADC	Analogue to Digital Converter
AOP	Acoustic Overload Point
API	Application Programming Interface
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
BOM	Bill of Materials
CNN	Convolutional Neural Network
CW	Continuous wave
DMM	Digital Multi-Meter
DSP	Digital Signal Processing
ECG	Electrocardiogram
ECM	Electret Condenser Microphones
EDA	Electronic Design Automation
EEPROM	Electrically Erasable Programable Read-Only Memory
FDM	Fused Deposition Modelling
FIR	Finite Impulse Response
FTP	File Transfer Protocol
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HHT	Hilbert Huang Transform
HTTP	Hypertext Transfer Protocol
I^2C/I2C	Inter-Integrated Circuit
I2S	Inter-Integrated Circuit Sound
IDE	Integrated Development Environment
IIR	Infinite Impulse Response
ILR	Implantable Loop Recorder
KNN	K-Nearest Neighbours
LDO	Low-Dropout Voltage Regulator
LDV	Laser Doppler Vibrometer
LIPO	Lithium Polymer
LMS	Least Mean Squares
MCU	Microcontroller Unit
MEMS	Micro-Electro-Mechanical Systems
ML	Machine Learning
MoSCoW	Must Should Could Would/Won’t
NICE	National Institute for Health and Care Excellence
NLMS	Normalised Least Mean Squares
NMF	Non-negative Matrix Factorisation
OP-AMP	Operational Amplifier
OS	Operating System
PCA	Principle Component Analysis
PCB	Printed Circuit Board
PCG	Phonocardiogram
PDM	Pulse Density Modulation
PETG	Polyethylene Terephthalate Glycol
PLA	Polylactic Acid
PPG	Photoplethysmography
PVA	Polyvinyl Acetate
RLS	Recursive Least Mean Squares
RNN	Recurrent Neural Network
RTOS	Real Time Operating System

SDK	Software Development Kit
SMD	Surface Mount Device
SNR	Signal to Noise Ratio
SOS	Second-Order Sections
SPI	Serial Peripheral Interface
SPIFFS	SPI Flash File Storage
SPL	Sound Pressure Level
STFT	Short Time Fourier Transform
SVM	Support Vector Machines
TDL	Tap Delay Line
TDM	Time Division Multiplexing
THD	Total Harmonic Distortion
TIC	Technology and Innovation Centre (University of Strathclyde)
TPE	Thermoplastic Elastomers
TPU	Thermoplastic Polyurethane
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
USB-C	Universal Serial Bus type C
VS Code	Visual Studio Code
WAV	Waveform Audio File Format (.wav)
WPANs	Wireless Personal Area Networks

1 INTRODUCTION

Cardiovascular diseases are the leading cause of death worldwide [1]. With a growing elderly population in the UK and half the total population predicted to develop a cardiovascular disease in their lifetime [2], there will be a major strain on the healthcare system in the future. It is estimated that costs for strokes in England will rise by 85% by 2050 [3]. Early detection and remote monitoring treatment are vital in managing these conditions before they worsen and require extensive hospital treatment, resulting in a high cost for both the NHS and patients.

Heart arrhythmias are a common indicator of many diseases and untreated/unmanaged arrhythmias put patients at risk for serious health complications e.g. strokes [1]. However, detecting and monitoring these arrhythmias requires clinical expertise and oftentimes further procedures. One of the most common diagnostic tools for arrhythmias is cardiac auscultation using a stethoscope – i.e. audibly listening for irregular heart sounds caused by the movement of blood through the heart valves and vessels. However, this requires the patient to be in the clinic, requires an arrhythmia to occur during consultation and requires extensive experience on the clinician's part to familiarize themselves with normal and abnormal heart sounds. As well as this, the interpretation of the heart sounds is completely subjective, and the heart sounds themselves are non-reproducible. Some arrhythmias such as ventricular tachycardia can be missed with the current auscultation system as they occur randomly for a few seconds, often asymptomatically, and can progress to a stage where an episode causes sudden cardiac arrest. With several types of arrhythmias that can range from harmless to life-threatening, or indicate other underlying diseases, being able to detect and distinguish them without requiring constant interaction with a medical professional can help many patients.

There are some viable alternatives used by hospitals currently for remote and continuous heart monitoring. These are all based on electrocardiogram (ECG) measurements, which record the electrical activity of the heart using electrodes attached to the skin. These devices range from traditional Holter monitors to physically invasive Implantable Loop Recorders (ILR) which are surgically placed underneath the skin. Typically, Holter monitors consist of several electrodes attached to the skin via a wet conductive gel. Recently there have been considerable advancements in portable ECG measurement devices to make them more user friendly and less bulky. Commercial options now offer devices available as a patch [4] or chest strap [5], as reusable or single use [4], able to record from 24 hours to 28 days with a number of electrodes required as low as one [6] and AI detection of abnormal heart sounds [7]. These advancements demonstrate a growing preference for accessible, non-invasive heart monitoring devices, as evidenced by a National Institute for Health and Care Excellence (NICE) investigation which found that a small wearable 14-day monitoring patch was cheaper, better at detecting abnormal heart rhythms, and preferred by patients over a traditional Holter monitor [8].

There is a growing trend in wearable health monitoring devices, such as fitness watches and mobile apps. For example, Apple Watch and a Samsung device can accurately measure a heart rate using photoplethysmography (PPG), an optical technique that detects changes in blood volume. However, the accuracy of these devices has been contested, especially as there are extreme variations in results due to thickness and colour of skin, age and non-uniform contact. While they are suitable for general fitness tracking, they do not have the accuracy or reliability required for medical use.

Various approaches have been developed to enhance the traditional stethoscope with advanced functionalities. StethoMe, for example, offers a remote digital stethoscope for home use with the ability to transmit recordings to a clinician immediately and utilizes Artificial Intelligence algorithms to classify lung sounds [9], while Tytocare offers clinician-guided remote auscultation for patients in quarantine [10]. These demonstrate the viability of a remote electronic stethoscope with either clinician expertise or computer-aided analysis.

Currently, there are no suitable wearable continuous heart sound sensing devices. In terms of wearable monitoring devices, there are plenty of ECG-based devices. However, ECGs and heart sounds offer a different perspective on heart function and an ECG is unable to provide information on problems arising from some structural abnormalities and valve irregularities. Additionally, ECG monitors, unlike heart sound monitors, require shaved skin, wet adhesive gels and to be kept away from strong magnetism. A suitable solution involves market options for a range of monitoring measurements, including heart sounds.

1.1 BACKGROUND THEORY

Auscultation is the act of listening to sounds from the heart, lungs, or other organs as part of a medical diagnosis. Heart sounds are measured at four major areas of auscultation: the pulmonary, mitral, tricuspid, and aortic valves, as indicated in Figure 1. These valves open and close depending on whether the cardiac cycle is in a systole or diastole state, and it is this process that produces the sounds for auscultation. The most significant sound point is S1, which occurs in the systole state just after the closing of the mitral and tricuspid valves. S2 is the next most significant, occurring at the start of diastole just after the closing of the aortic and pulmonic valves. Finally, S3 and S4 are abnormal sounds that can occur during diastole.

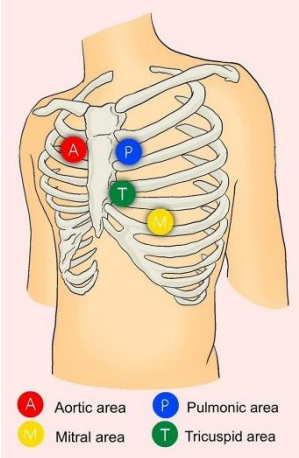


Figure 1. Diagram of heart auscultation zones

1.2 LITERATURE REVIEW

The development of wireless cardiac auscultation systems has evolved significantly over the last two decades. This section reviews the existing body of work in areas relevant to the proposed solution, including wireless transmission technologies, auscultation techniques, signal processing methods, artificial intelligence applications, and user studies in e-health.

Remote heart sound monitoring has been demonstrated in several studies. Haoran Ren, et al in [11] developed a digital stethoscope that includes a mobile application for storing and displaying recorded sounds and achieved an accuracy of 86.66% in detecting abnormal heart sounds. [12] developed a wearable auscultation device with the option for either user-initiated recordings or continuous ten second recordings at ten-minute intervals with BLE data transfer and verified suitable recording collection and transmission by comparing heart sound signals to a simultaneous ECG. [13] and [14] provide foundational overviews, identifying typical heart sound frequencies (e.g. S1 and S2 located in 10-200 Hz range, murmurs up to 420 Hz). Wearable implementations have grown in sophistication. Studies such as [15] and [16] highlight integration of soft materials, motion artifact mitigation and low-power design. Other efforts, like [17], focus on performance in high-noise environments, using

microphone arrays, physical modelling and statistical analysis (e.g. ANOVA) to evaluate and reduce interference.

Sound sensors can be susceptible to noise especially due to improper contact between the skin and the sensor and impedance mismatch. Studies investigating the mechanical coupling between the skin and the sensor include [15] where a soft, flexible silicone-based design had reduced motion artifacts compared to a rigid design due to its ability to conform to the skin and minimise the air-gap. It also features an air cavity which helps record heart sounds correctly even in noisy conditions. [18] tests different combinations of water and hydrogels enclosed in silicone as a propagation medium to record heart sounds when placed over clothing or hair.

Early work explored various wireless methods for medical telemetry in auscultation. Zigbee was evaluated for phonocardiograph signal transmission in [19], but was found to be unsuitable for auscultation data due to bandwidth limitations. WLAN MAC schemes aimed at low-latency applications were studied in [20], although not all health applications require real-time responsiveness. BLE and mobile networks have gradually replaced older systems. For instance, [11] presents BLE specifications for wireless auscultation, while [21] reflects earlier Global System for Mobile communications (GSM)-based approaches. Broader technological shifts were predicted in [22], a paper written in 2004, which anticipated the central role of Bluetooth , Wi-Fi and Zigbee in wireless medical systems.

Signal analysis plays a crucial role in auscultation-based diagnosis. [23] and [24] discuss various time-frequency analysis tools, including wavelet transforms, Hilbert-Huang Transforms (HHT), Mel-Frequency Cepstral Coefficients and statistical models like Multivariate Autoregressive and Univariate Auto Regressive models. Papers, like [11] compare the limitations and advantages of different analysis methods, including STFT, Wigner-Wille and HHT. Filtering approaches for noise reduction are discussed in [25], a paper from 1998, which utilised Least Mean Squares (LMS) and Normalised LMS (NLMS) adaptive algorithms for performance evaluation in noisy scenarios.

Several studies have evaluated user and clinical acceptance of remote auscultation systems. [26] assessed the feasibility of digital auscultation during quarantine and found it acceptable across 250 patients. Similarly, [27] highlighted the potential of self-monitoring personalisation through signal profiles and early intervention. A survey assessing patient confidence in remote, self-operated auscultation found that 85.6% of participants partially or fully trusted themselves to collect reliable data and 80% had prior experience with home-monitoring systems [28]. The main concern was obtaining incorrect results due to incorrect measurements or improper use, emphasizing the need for reliability and ease of use. User acceptance studies like [29] report positive attitudes toward wearable health monitoring, particularly among aging populations. The studies underscore the role of such systems in preventive care and healthcare cost reduction.

Machine learning methods are increasingly employed in diagnostic systems. [30] demonstrated the use of radial basis Support Vector Machines (SVM) and Principle Component Analysis (PCA) for heart sound classification, while [15] employed deep learning in the form of a Convolutional Neural Network (CNN) to reduce motion artifacts and improve classification. [31] offers insights into commercial applications in digital stethoscopes. [32], a mobile cardiac monitoring system achieved high diagnostic accuracy using classical detection algorithms for arrhythmias and myocardial infarction.

As wireless systems evolve, security remains a concern but [33] explains how resource constraints in wireless sensor networks limit encryption options, especially in high-frequency data transmission scenarios.

The reviewed literature demonstrates a broad trajectory of innovation in wireless auscultation and health monitoring, from early GSM and Zigbee-based systems to modern AI-enhanced, BLE-enabled wearable devices. Developments in signal processing, filtering and user acceptance have laid strong foundation for home based and mobile health monitoring solutions.

1.3 PROJECT DESCRIPTION

This project aims to create a wearable, continuous monitoring system to record the heart sounds of the wearer, while being smaller and cheaper than a portable ECG device, and less invasive than an ILR. It will pair to a mobile app providing instructions of use, processing of the recorded sounds, and feedback to the patient. To eliminate errors introduced through clinician subjectivity and to prevent overwhelming healthcare systems ill-equipped to handle large incoming data streams, this system should be able to analyse and identify if a heart arrhythmia is experienced.

The original project proposal was for a 'Wireless Cardiac and Pulmonary Auscultation System' and took inspiration from the existing 'StethoMe' product – a handheld, patient-operated device for pulmonary recordings [34]. From this, the focus of the project was to design a similar device, but specifically for long-term continuous at-home cardiac auscultation and monitoring. The initial system design was based on early discussions with the project proposer and similar successful projects as identified through the Literature Review (Section 1.2). For measurement of heart sounds, it was recommended to use a type of contact transducer. To make the device easier for the patient to use at home, guiding the system with a mobile application was identified as a requirement, which would also require any analogue measurements to be converted into digital for processing and classification. It was highlighted by the project proposer that power consumption would be a concern; therefore, choice of Microcontroller Unit (MCU) and transmission protocol would be important (discussed further in Sections 2.1 and 2.2) as well as low-power modes (e.g. continuous sampling vs standby).

The system will consist of two key parts, as shown in Figure 2: the hardware elements (including circuitry, shoulder brace and housing), and the software element. The physical aspect will include an adjustable brace to hold the device in the correct position and house all the electronic components. It will also include a contact adhesive that will prevent the listening device from slipping and reduce noise from external parameters. The circuitry will consist of a MEMS microphone (to record the heart sounds), an MCU with transmission capability and an internal Analogue to Digital Converter (ADC), memory to temporarily store recordings when connection isn't available, and a 3.7V battery, with progressive integration of 1st and 2nd stage software filtering. The hardware must strongly focus on Signal to Noise Ratio (SNR) as SNR must be positive when noise is at least 40dB (roughly urban ambient noise level).

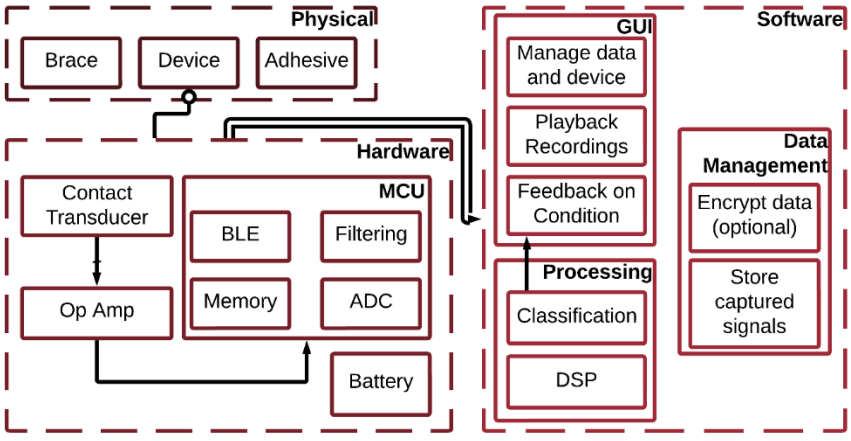


Figure 2. System architecture diagram

The software aspect will have three key elements: data processing, data management, and user interaction. The data processing will include Digital Signal Processing (DSP) for feature extraction and heart sound isolation. It will then make use of a Machine Learning model in the form of a Convolutional Neural Network (CNN) to detect and classify any arrhythmias, discussed further in Section 6. The model will be trained on open-source datasets that reflect similar data points/features as the device output, split 80-10-10 for Training-Validation-Testing. It will be trained on a device with a high computation capacity, but the model will run on the patient's app.

The data management aspect will require the recorded and processed sounds to be logged on the patient's phone and catalogued by date and/or captured arrhythmia detected. There is also the optional criteria to encrypt the patient's data. The last software aspect is the Graphical User Interface (GUI). This will present instructions of use to the patient, allow them to listen to/visualise the recorded heart sounds, and give them feedback on their condition – i.e. display an alert if an arrhythmia was detected and suggest seeking medical attention if necessary. The progression of using the system is illustrated in Figure 3.

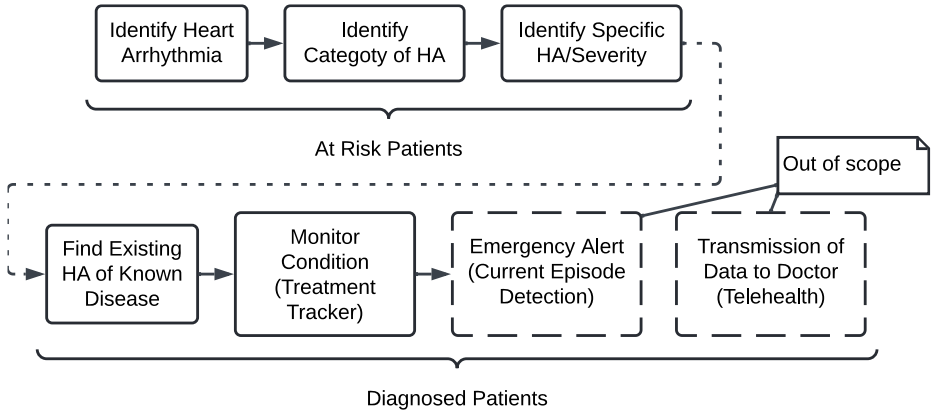


Figure 3. System use flow chart

The device will be evaluated by simulating a range of heartbeats through a speaker covered with a substrate that replicates the acoustic properties of human tissue. This test will be conducted in both a quiet room and a noisy environment to determine the device's suitability for everyday use. Further detail on test strategy discussed in Section 9.1.

It should also be noted, for privacy and security, that the device will not be recording at a frequency that could capture speech. Furthermore, at no point will the system send data to any third parties or any external locations (other than the user's phone).

2 HARDWARE SELECTION

2.1 TRANSMISSION PROTOCOL

In modern healthcare systems, reliable and energy-efficient data transmission is critical, particularly for wearable devices. These systems require protocols that not only ensure low energy consumption but also maintain high reliability and seamless connectivity. Upon reviewing the literature, several protocols stood out as potential solutions for low-energy data transmission.

2.1.1 IEEE 802.15.4

The IEEE 802.15.4 standard is a foundational technology for low-power, low-data-rate Wireless Personal Area Networks (WPANs). It is especially suitable for sensor networks used in healthcare monitoring systems due to its low complexity and extended battery life. However, its low data throughput limits its use to applications that transmit small packets of data only and is also susceptible to interference from Bluetooth and Wi-Fi [35].

2.1.2 Zigbee

Built on the IEEE 802.15.4, Zigbee poses a promising solution as a transmission protocol in healthcare systems due to its excellent power efficiency [36], [37]. It relies on a Zigbee gateway to communicate with external networks, which make it well-suited for fixed environments like care homes or patients spending most of their time in a single location (e.g. home). This dependency can limit patient mobility.

2.1.3 Wi-Fi and 5G

Wi-Fi and 5G are both high-speed transmission protocols that support large data volumes and are widely used in healthcare settings for applications like telemedicine, medical imaging, and real-time patient monitoring. Their ability to provide high-bandwidth connectivity make them indispensable in hospital environments where continuous data streaming and integration with electronic record systems are necessary. However, both Wi-Fi and 5G require significantpower which limits their use in battery-dependent portable monitoring devices.

2.1.4 Bluetooth Low Energy (BLE)

Bluetooth is one of the most widely adopted wireless protocols, offering a balance between power efficiency and data transmission speed. Advancements like Bluetooth Low Energy as part of the Bluetooth 5.0 protocol have made it an attractive option for healthcare applications as it is specifically optimized for low-power operation over long time periods [12], [36], [11]. Its short-range communication – at about 10 meters – is sufficient for wearable sensors and portable monitoring devices but less suitable for larger scale (e.g. healthcare equipment in a hospital). For these reasons, BLE was selected as the transmission protocol in this project.

2.2 MICROCONTROLLER UNIT (MCU) SELECTION

For MCU selection, several criteria must be carefully evaluated to meet the specific needs of the project. These include processing power, sampling rates, energy efficiency, memory requirements, peripheral support, and connectivity capabilities.

2.2.1 Evaluation Criteria

Low-power microcontrollers are essential to ensure prolonged operation of wearable healthcare devices without frequent recharging or battery replacement. Features like sleep modes, low-power peripherals, and dynamic voltage scaling play a significant role in optimizing power consumption.

Generally, devices like ECG monitors require processing of biomedical signals in real time. In the proposed system the algorithmic processing will be offloaded to the phone, so the MCU will only have

to transmit the sampled data. Given the importance of transmission protocols in this application, microcontrollers with built-in communication modules were preferred. To enable the MCU to store some samples on-board, options with EEPROM/flash were preferred.

To enable the MCU to interface with microphones, sufficient pin peripheral support was also required – including Analogue to Digital Converters (ADCs) for analogue microphones and I^2C/SPI interfaces for digital ones. MCUs with a low development-kit cost, as well as a low Surface Mount Device (SMD) cost, were also preferred along with capability to be coded in common languages like C and Python for ease of development.

2.2.2 Options and Selection

A range of boards from a range of manufacturers were considered. Table 1 summarises the comparison of these constraints across manufacturers*.

Table 1. Summary of MCU options

Manufacturer	Key strengths	Potential Drawbacks
Espressif (ESP32)	Affordable overall, small, multiple communication protocols, IDE versatility, works with C/Python, RISC-V based, very well documented.	RISC-V less power-efficient compared to ARM.
Nordic	Power efficient (ARM-based), integrated BLE, own IDE support.	Dev boards are relatively expensive, limited additional communication protocols.
Texas Instruments	Power efficient (ARM-based), affordable, multiple upload options (IDEs and web-interface), onboard debugging.	Limited functionality and peripheral support.
STMicroelectronics	Power efficient (ARM-based), multiple communication protocols, affordable SMDs, works with Python.	Developments boards are on the pricier side.
Renesas	Power efficient (ARM-based), affordable SMDs, own IDE support.	Limited communication protocol support, pricier development kits, lack of prior working familiarity.
Silicon Labs	Power efficient (ARM-based), IDE versatility, affordable SMDs.	Limited communication protocol support, pricier development boards, limited familiarity with the ecosystem.
Cypress (Infineon)	Power efficient (ARM-based), own IDE, decently priced dev boards.	SMD pricing somewhat high, limited communication protocols support.
Apollo (Ambiq)	Extremely power-efficient (ARM-based), affordable SMDs.	Very expensive dev boards, less familiarity with the ecosystem.
ONSemi	Power efficient (ARM-based), affordable overall.	Unclear comms protocol diversity.
NXP	Power efficient (ARM-based), IDE versatility, supports C and Python.	Pricier dev boards (potential for access through academic resources)
Ezurio (Laird)	Affordable SMD, multiple comm protocols.	Expensive development boards, less familiar ecosystem.

*Although Qualcomm are widely known, they were not considered in the selection process due to the current ARM lawsuit and overall difficulty of purchase.

From the above analysis, the best candidates emerged as Espressif's ESP32 C3 and C6, Nordic Semiconductor's nRF5340, and Apollo's (Ambiq) Apollo3 Blue. Espressif's ESP32-C6 was selected due to the project group's previous experience and familiarity with the device, as well as the low cost of SMDs and development-kits.

2.3 OPERATING SYSTEM (OS)

The choice of an Operating System (OS) was a critical step as it determined the runtime environment for the application and the overall development workflow, compatibility, and maintainability. Below is an analysis of the OS options evaluated, including their strengths, limitations, and overall suitability.

2.3.1 Evaluation criteria

The OS must contribute to the ease of development by ensuring there is available and detailed documentation, tools, and community support, as well as support for real-time processing, multi-threading (to fully utilise the MCU) and system-level features like POSIX compatibility (APIs). Lastly, the cost and licensing were considered, with free and/or open-source solutions preferred. An overview of the evaluated OS's is summarised in Table 2.

Table 2. Overview of Evaluated Operating Systems (OS).

Operating System	Strengths	Limitations	Verdict
Contiki-NG	-	For networked projects, docker-based development	Poor fit for proposed application - PASS
openERIKA	-	Paid Licensing	PASS
MicroC/OS (MicriumOS)	Reliable RTOS for embedded systems	Limited resources and documentation	PASS
NuttX	POSIX-compliant, broad peripheral support	Requires KConfig, steep learning curve	PASS
RIOT OS	Lightweight, GNU Make-based, good documentation	Limited advanced features compared to alternatives	Considering
Zephyr OS	Advanced RTOS, promising features	Python-based development workflow required	Considering
ChibiOS/RT	Lightweight RTOS, energy efficient	Requires special IDE for development	PASS
FreeRTOS	Widely used, robust real-time features, partial POSIX support, multi-variant	Vanilla SMP has limited multi-core support	Leaning towards

Based on this analysis, FreeRTOS was selected.

2.4 MEMORY FOR SAMPLE-STORING

To ensure reliable data collection and retention for transmission, incorporating memory for storing audio samples is essential. Memory serves as a buffer and long-term storage solution, allowing collected samples to remain even if the system temporarily loses power or signal. There are two commonly used types of non-volatile memory suitable for his purpose, Electrically Erasable Programmable Read-Only Memory (EEPROM) and Flash memory.

2.4.1 EEPROM

EEPROM provides fine-grained access of memory control, allowing individual bytes to be re-written and/or erased through program control, making it suitable for applications with frequent and small updates. Compared to Flash, EEPROM typically has smaller storage capabilities and slower write speeds, which may be a limiting factor for high-volume audio sample storage.

2.4.2 Flash memory

Flash memory offers larger storage capacities and faster write speeds, making it more suitable for storing a continuous stream of audio samples whilst supporting a higher read/write operation cycle count. Flash writes and erases data in blocks rather than individual bytes. Given the requirements for this project, flash memory is the superior choice for storing samples due to its higher capacity and faster performance.

2.4.3 SD Card

Using an SD Card was considered. This would allow the device to store much larger amounts of data. However, reading and writing to the SD card would also be more time and energy intensive. It would also require significantly more space, increasing the footprint of the PCB. For the current design purposes this was not preferred. Additionally, due the size of the recorded .wav files, larger storage was not required.

2.4.4 Memory calculations for standby time

To analyse the performance of the ESP32-C6 MINI H8, namely how long it can store data before it would need to be transferred, the below calculations were performed. The WAV file itself contains information inside a 44-byte header. The total memory requirement for this array can be calculated based on the sampling configuration required.

For example, for a sample rate of 1 kHz and a total duration of 10 seconds, the resulting samples can be calculated as:

$$samples = 1000 \times 10 = 10\ ksamples \tag{2.1}$$

Three common ways for storing wav samples include floating point, 32-bit integers, and 16-bit integers. Their respective storage requirements are shown in Table 3.

Table 3. Memory calculations for each sample type.

Type	bytes/sample	Total file size – kilobytes (decimal)	Total file size – kibibytes (binary)
float	4	40 KB	39.06 kiB
int32/int	4	40 KB	39.06 kiB
int16	2	20 KB	19.53 kiB

For the memory allocated specifically for file storage, the device can store:

$$number\ of\ wav\ files = \frac{partition\ size}{bytes\ per\ file} \tag{2.2}$$

Where for float, int32 and int :

$$bytes\ per\ file: 40{,}000 + 44 = 40{,}044\ bytes\ per\ file \tag{2.3}$$

And for int16:

$$bytes\ per\ file: 20{,}000 + 44 = 20{,}044\ bytes\ per\ file \tag{2.4}$$

Therefore, (2.2) becomes:

$$number\ of\ wav\ files = \frac{7{,}274{,}496}{40{,}044} = 181.66\ or\ 181\ files \tag{2.5}$$

Or

$$number\ of\ wav\ files = \frac{7{,}274{,}496}{20{,}044} = 362.93\ or\ 362\ files \tag{2.6}$$

This means 181 or 362 audio files can be stored before the device runs out of memory, depending on the data type selected.

This is equivalent to:

$$181 \times 10\ seconds \times \frac{1\ hour}{3600\ seconds} \approx 0.50\ hours \tag{2.7}$$

Or

$$632 \times 10\ seconds \times \frac{1\ hour}{3600\ seconds} \approx 1\ hour \tag{2.8}$$

This means 30 minutes or 1 hour of data can be stored before needing to transfer/overwrite the current data, again depending on the data type selected.

2.5 MICROPHONE

Microphone selection is a critical aspect of the proposed solution. Various types of microphones – categorized by technology and interface in Sections 2.5.1 and 2.5.2 respectively – were evaluated based on factors such as Acoustic Overload Point (AOP) in Sound Pressure Level (SPL), frequency response, power consumption, and integration complexity.

2.5.1 Microphone Technologies

2.5.1.1 Micro-Electro-Mechanical Systems (MEMS)

MEMS microphones consist of a tiny diaphragm and a backplate that form a variable capacitor. They are either omnidirectional or unidirectional and typically have a high AOP around 120-135 dB SPL, making them suitable for noisy environments. They also have a higher SNR compared to ECM or piezo, providing clean audio output with low background noise. MEMS are compact, robust and highly sensitive, and their small size makes them ideal for this project.

2.5.1.2 Electret Condenser Microphones (ECM)

ECMs use a permanently charged electret material to maintain a fixed electric field, known for their high sensitivity and broad frequency response. ECMs require a power supply (bias voltage) for their operation. Most are omnidirectional, but some (e.g. cardioid) are unidirectional for applications

requiring focused sound pickup. They typically have lower AOPs compared to MEMS, making them suited for moderate pickup of sound signal in noise, but less capable in stronger sound pressure conditions. Although slightly larger than MEMS microphones in package size, they are a cost-effective solution for general-purpose acoustic sensing.

2.5.1.3 Piezoelectric Microphones

Piezoelectric elements can function as microphones. The piezoelectric materials generate voltage when mechanically stressed. They are generally omnidirectional but may exhibit directional characteristics if encased in a physical housing. Their AOPs and SNRs are lower in comparison to MEMS and ECMs, resulting in noisier output.

2.5.2 Analogue vs. Digital Microphones

2.5.2.1 Analogue Microphone Packages

Analogue microphones output raw analogue signals, typically requiring external filtering and amplification. They offer high fidelity and are generally simpler to integrate in low-latency systems. However, the system design must be adapted to accommodate additional components such as a hardware ADC, a 1st stage filter to remove DC component and minimise electrical noise, an Operational Amplifier (OP-AMP) to amplify the signal, and then a 2nd stage filter to remove the added noise floor introduced from the OP-AMP.

2.5.2.2 Digital Microphone Packages

These typically have built-in ADCs, filters, OP-AMPs and output data in formats like I2S, PDM, or TDM, simplifying the design of digital processing systems. However, they can introduce latency and have slightly higher power consumption. Also given the ADC, filtering and amplification is done in-package, there is typically very little control over these parameters.

2.6 COMPONENT SELECTIONS

Table 4 (following page) lists each selected component from the above discussions. Their respective pricing and overall budgeting are explained in Section **Error! Reference source not found.**. As shown, a variety of microphones were ordered to test in-house the results are described in Section 4.1.

Table 4. Component selection, quantity and reasoning.

Item Description	Model	Quantity	Purpose
Microcontroller	ESP32-C6	3	Coding
External 18-bit ADC 4-channel with programmable gain	MCP3224	1	Testing the effect of external ADC and amp on signal quality and resolution
I2S MEMS digital microphone	SEN0526	1	Testing the suitability of a digital microphone
Analogue MEMS microphone	SEN0487	1	Testing the suitability of an analogue microphone
ECM microphone	ADA1064	1	Testing the suitability of an ECM omnidirectional microphone
ECM microphone	ADA1935	1	Testing how the size impacts the sound pickup, compared to the other ECM
ECM cardioid microphone	PUM-5250L-C3310	1	Testing the suitability of a microphone from literature [38]
ECM literature microphone	CMI-5247TF-K	1	Testing the suitability of a cardioid ECM

3 SOFTWARE SELECTION

3.1 MACHINE LEARNING MODEL

3.1.1 Selection of Model

Machine Learning (ML) models are software structures that use large datasets and pre-defined algorithms to gradually improve AI-based searches over time, until the model can classify data items without human intervention. Therefore, for the identification of anomalies/arrythmias in the cardiac cycle, the proposed solution is to train an ML model on phonocardiogram datasets to automatically classify the heart sounds recorded by the device.

The three main model categories are supervised, unsupervised and semi-supervised. Supervised involves using pre-labelled datasets to adjust the weights of model parameters until it finds an appropriate fit, while unsupervised involves the analysis of unlabelled datasets to detect patterns and sequences with no human intervention. Semi-supervised represents a middle-ground between the two methods most useful when there is only a small training dataset available [39].

To keep the project in scope, the ML model aimed to classify heart anomalies in stages, working down the classification layers to narrow the result – therefore, the datasets the ML model was trained on needed to have one label for each, as classified by an expert. The most appropriate choice for achieving this was a supervised model.

3.1.2 Selection of Algorithm

A range of algorithms that were suitable for supervised learning models and appeared often in literature for similar investigations [40] were considered. The results are shown in Table 5.

Table 5. ML algorithm selection

Algorithm	Description	Data Format		Problem Application
		Input	Output	
Logistic Regression	Estimate discrete (binary) values based on a given set of independent variables by fitting to a logistic function.	Discrete values	Probability value between 0 and 1	Classification
Linear Regression	Estimate real values based on continuous variables by a line of best fit.	Real values	Linear equation for line of best fit	Regression, Predictor
Neural Networks	Classifies by passing data through different layers of the network depending on whether it reaches a certain threshold value	Tensors – data points encoded as vectors	Set no. of nodes corresponding to no. of classes	Classification
K-Nearest Neighbours (KNN)	Classify data points by distance between point and nearby cluster	Real values	Property value for a certain object – avg. of values of nearest neighbours	Classification
Decision Trees	Population split into distinct groups, continuously broken down into subsets	Most significant attributes of dataset	Best fit group	Classification, Regression

The best options were Neural Networks or Decision Trees, as both are suited to classification problems, output a specific decision, and can use techniques such as batch normalisation to generalise data and classify only by the most significant attributes – e.g. the frequency bands that are repeated the most, or stand out as abnormal within a heart sound. Since the device recordings will be converted to digital

signals, greater flexibility would be gained from representing key points as vectors. Therefore, the best choice was a neural network.

The two main kinds of neural network are Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN). While similar, RNNs (in a general sense) learn to recognise patterns across time, while CNNs learn to recognise patterns across space. RNNs classify based on sequential patterns (i.e. preceding and following data points), feeding previous layers in the network as inputs into the next. CNNs look for data points, then classify based on the combination of these into larger structures. CNNs are therefore better suited to looking for the same pattern over different spaces and records (e.g. a heart anomaly over a spectrogram), which made a CNN a better choice.

3.1.3 Selection of Dataset

The Classification of Heart Sound Recordings database (2016) contains a total of 3126 heart sound recordings at all four auscultation locations and includes a validation test suite with 300 entries. Using a standard 80-10-10 data split for training-testing-validation:

- Training = 80% = 2500 recordings
- Testing = 10% = 313 recordings
- Validation = 10% = 313 recordings

3.2 APP DEVELOPMENT

It was decided to make an application specifically for Android operating systems, rather than Apple (or both) as Apple is notorious for being difficult to get permissions to test any app on their devices. Android Studio was selected to develop the mobile application as it is the current industry standard and because some group members have previous experience.

Java was chosen as the programming language rather than the mobile specific language, Kotlin, due to the group having significant experience with Java but none with Kotlin. Learning Kotlin on top of the learning curve of using Android Studio was not feasible within the project timeline.

3.3 MICROCONTROLLER PROGRAMMING

Due to the selection of an ESP32 microcontroller, Espressif's IoT Development Framework (IDF) was the best choice to compile and program the chip. The IDF also natively offers FreeRTOS functionality.

Visual Studio Code (VS Code) was selected as the development environment with the Espressif IDF add-on. Choosing to use the VS Code add-on instead of downloading the IDF separately means that the majority of the tool's configuration is handled by VS Code and provides quality of life shortcuts for easier communication with the microcontroller. C was chosen as the programming language for the microcontroller firmware due to the language's low overhead, making the code both power and memory efficient.

The onboard USB-C port will be used to program the ESP32-C6 Mini H8 MCU via USB Device Firmware Upgrade (DFU mode) and to communicate with the host system over the USB Communication Device Class (CDC) interface. This approach eliminates the need for a dedicated USB-to-serial converter by leveraging the ESP32-C6's native USB peripheral. Firmware flashing and serial monitoring are handled directly over the USB-C connection using *idf.py* with the appropriate DFU drivers. The decision to avoid external debugging/ flashing tools such as the ESP-JTAG board was made to minimize cost, since the built-in USB CDC interface was sufficient.

To enter CDC, a combination of buttons (BOOT, RESET) had to be held and released.

3.4 CODE MANAGEMENT

The entire codebase was managed using Git and is available on GitLab and GitHub under public domain, which not only tracks changes over time but also ensures that any version of the code used to produce results is available for future reference. This allows the general public to recreate the same device with minimal discrepancies. The repositories are under AGPL-3.0 license.

4 ELECTRONICS

4.1 COMPONENT TESTING

The component testing phase focused on evaluating the performance of various microphone technologies, including the MEMS, ECMs, and piezoelectric buzzers. This testing was conducted using the DSO138 portable oscilloscope kit.

The primary objectives of this testing phase were:

I. To verify the functionality and responsiveness of each component.
II. To measure and compare the output signal characteristics (including voltage amplitude and waveform integrity) under controlled acoustic excitation.
III. To establish a baseline understanding of the signal quality each component produces for subsequent analysis in the project (short-list the components to focus on in further testing).

4.1.1 Testing Methodology

Each microphone was tested under identical conditions to ensure consistent and comparable results. The components were connected to the DSO138 oscilloscope using a breadboard circuit and necessary interface circuitry according to their datasheet's requirements (e.g. pull-up resistors, biasing capacitors or impedance matching). The oscilloscope's captured waveform shapes and manual peak-to-peak voltage and noise levels were recorded.

4.1.2 Results

From the testing carried out, the MEMS microphones seemed to outperform the other microphones, specifically the SEN0487 analogue MEMS microphone. Key observations are summarised in Table 6 (following page), while a detailed analysis is available in the sections below.

4.1.2.1 MEMS

The SEN0487 and SEN0526 analogue MEMS managed to perform very well, given their directionality, high AOP and sensitivity. Their responses are illustrated in Figure 4a and Figure 4b, respectively.

4.1.2.2 ECM

To test ECM microphones, 5 models were obtained through ThePiHut, Mouser Electronics, and university resources: PUM-5250, CMI-5247, ADA-1935, CMA-4544PF and R-TECH 350095. Out of all the models, the SMD PUM-5250 (also seen in the literature [41]) performed the best, followed by CMI 5247 and R-TECH 350095. They all suffered from background noise and transient response and are of various sensitivities. They are shown in Figure 4c-h.

4.1.2.3 Piezoelectric Buzzers

To test piezoelectric buzzers as microphones (essentially reversing their function), three buzzers were obtained through university resources, a generic RS PRO piezo audio indicator, the Toko PB2720, and a generic piezo buzzer. The first two were enclosed buzzers and their respective responses are shown in Figure 4h-i. The non-enclosed generic piezo buzzer's response is shown in Figure 4j.

4.1.2.4 Challenges and limitations

Testing was limited by the resolution and bandwidth of the DSO138, which may not fully capture high-frequency details or subtle waveform change. Additionally, interfacing challenges, such as impedance mismatches for the ECM microphones, internal impedance of the breadboard, and wire impedance, all contributed to non-ideal results. An attempt was made using Veroboard, which helped in minimising internal impedance, but the results were still were not ideal.

Table 6. Observations from testing microphones.

Model number	Microphone technology	Sensitivity	Noise levels	Extra circuit complexity	Notes
SEN0487	MEMS	Very High	Very Low	Not required	Excellent
SEN0526	MEMS	High	Low	Not required	Excellent
PUM 5250	ECM	High	High	Somewhat complex	Response uniformity, Transient response
CMI 5247	ECM	Medium	High	Somewhat complex	Transient Response
ADA 1935	ECM	Medium	High	Somewhat complex	Transient Response
R-Tech 350095	ECM	Low	Medium	Somewhat complex	Transient Response
CMA 4544PF	ECM	Low	High	Somewhat complex	Transient Response
RS PRO piezo audio indicator	Piezoelectric Buzzer	Low	High	Not required	Low-frequency limitation, enclosed
Toko PB2720	Piezoelectric Buzzer	Very Low	Very High	Not required	Low-frequency limitation, enclosed
Generic piezo buzzer	Piezoelectric Buzzer	Low	Medium	Not required	Low-frequency limitation

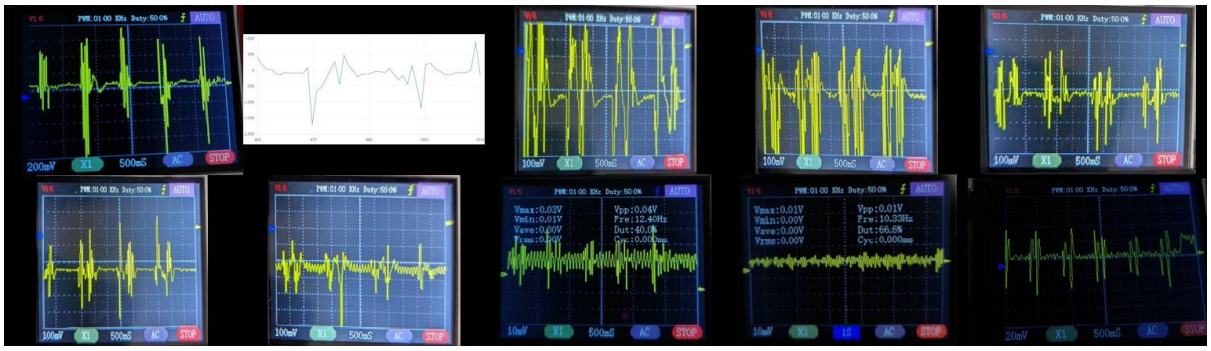


Figure 4. a) SEN0487 Analogue MEMS b) SEN0526 Digital MEMS c) PUM-5250 ECM d) CMI-5347 ECM e) ADA 1935 ECM f) R-TECH 350095 ECM g) CMA-4544PF h) RS-PRO piezo audio indicator i) Toko PB2720 piezo buzzer j) Generic piezo buzzer

4.1.2.5 Findings

The testing phase provided valuable insights into the performance characteristics of each microphone technology. MEMS were selected - given their superior performance and no extra circuitry requirements. Further refinement of the testing setup will be required to explore specific aspects in detail, such as dynamic range and current draw.

4.2 DIGITAL SIGNAL PROCESSING (DSP)

4.2.1 Adaptive Filtering

To ensure the captured audio samples are accurate and as free from noise (electrical, thermal or environmental) or interference (due to multiple transducer sampling) as possible, digital filtering is a critical step. The proposed solution aims to employ real-time DSP adaptive filter algorithms to handle noise suppression, improve signal quality and enhance system storage usage (coefficient storing). This is to allow adaptability in the digital filter as opposed to a physical filter embedded on the device. It also allows the device to be smaller and less complex in terms of circuitry. The motivation to perform the filtering on the ESP32 rather than the phone is because the BLE transmission may corrupt the signal – given that no redundancy was added, making recovery very difficult.

4.2.1.1 Wiener Filter Theory

The Wiener filter seeks to find a linear filter that produces an estimate of a desired signal by minimising the Mean Square Error (MSE) between the estimated and desired signals.

Given :

- Input signal: $x(n)$
- Desired signal: $d(n)$
- Filter coefficients: $w = [w_0, w_1, \dots, w_{M-1}]^T$ (where T denotes the transpose of the matrix)

The estimated signal is:

$$\hat{\mathrm{d}}(\mathrm{n}) = \sum_{\mathrm{i}=0}^{\mathrm{M}-1} \mathrm{w_i x(n-i)} = \mathrm{w^T x(n)} \tag{4.1}$$

With the error being calculated as:

$$\text{error: } \mathrm{e(n)} = \mathrm{d(n)} - \hat{\mathrm{d}}(\mathrm{n}) \tag{4.2}$$

With the cost function being defined as:

$$\xi = \mathcal{E}\{|\mathrm{e(n)}|^2\} \tag{4.3}$$

Where:

- $\mathcal{E}\{\}$ denotes the expectation of a function, with a finite number of outcomes as a weighted average of all possible outcomes. In the case of a continuum of possible outcomes, the expectation is defined as by integration.

And by substituting (4.2) into (4.3):

$$\xi = \mathcal{E}\left\{\left|\mathrm{d(n)} - \mathrm{w^T x(n)}\right|^2\right\} \tag{4.4}$$

Minimising the cost function leads to the Wiener-Hopf equations:

$$\mathrm{R w_{opt}} = \mathrm{p} \tag{4.5}$$

Where:

- $R = \mathcal{E}\{x(n)x^T(n)\}$ is the autocorrelation matrix
- $p = \mathcal{E}\{x(n)d(n)\}$ is the cross-correlation vector

If R is not invertible, being a singular matrix due to correlated input (like sound), the pseudoinverse is used:

$$w_{opt} = R^{+}p \tag{4.6}$$

The Wiener-Hopf solution is optimal in the calculation of MSE but requires prior knowledge of statistics and has a high computational cost.

4.2.1.2 Least Mean Square (LMS)

LMS is an iterative Stochastic Gradient Descent (SGD) approximation of the Wiener solution:

$$w(n+1) = w(n) + \mu e(n)x(n) \tag{4.7}$$

Where:

- μ is the step size controlling the convergence speed and stability
- $e(n) = d(n) - w^{T}(n)x(n)$

The LMS has a low computational cost, (O(L)), {where L is number of multiply-accumulate (MAC) operations}, and is simpler in design. It is however slow to converge – especially for correlated inputs - and its performance is sensitive to step size μ.

4.2.1.3 Normalised Least Mean Square (NLMS)

The NLMS improves upon LMS by normalising the step size:

$$w(n+1) = w(n) + \frac{\mu}{||x(n)||^{2} + \delta} e(n)x(n) \tag{4.8}$$

Where:

- δ is a small value to prevent division by zero

This normalisation prevents the algorithms from making large updates when the input signal has high energy, improving stability and convergence.

NLMS is assisted by an adaptive step size to improve convergence and is better than LMS for correlated signals but has a slightly higher computational cost.

4.2.1.4 Recursive Least Squares (RLS)

RLS is an adaptive filtering algorithm that uses recursive matrix algebra to update filter weights exactly and minimizes a different cost function, namely the weighted least-squares defined as:

$$\xi(n) = \sum_{i=0}^{n} \lambda^{n-i} \left| d(i) - w^{T}(i)x(i) \right|^{2} \tag{4.9}$$

Where:

- λ is the forgetting factor ($0 < \lambda \leq 1$) which determines how much past errors are taken into account for the calculations

RLS uses:

- a gain vector k(n) defined as:

$$k(n) = \frac{P(n-1)}{\lambda + x^{T}(n)P(n-1)x(n)} \tag{4.10}$$

- The estimation error, e(n) defined as previously:

$$e(n) = d(n) - w^{T}(n-1)x(n) \tag{4.11}$$

- The weight update as:

$$w(n) = w(n-1) + k(n)e(n) \tag{4.12}$$

Where the weights are adjusted using the gain vector scaled by the estimation error.

- And the covariance matrix update P(n) as:

$$P(n) = \frac{1}{\lambda}[P(n-1) - k(n)x^T(n)P(n-1) \tag{4.13}$$

Where this step updates the inverse correlation matrix efficiently – no explicit matrix inversion needed at each time step.

RLS is quicker to converge, compared to the aforementioned algorithms, especially when the input is correlated and tracks time-varying systems more precisely, being the exact solution to the exponentially weighted least squares problem. However, it is more complex due to the computational cost of $O(L^2)$ as it involves matrix-vector multiplications and updates at each step, as well as needing more memory to store and update the (LxL) matrix P(n).

4.2.1.5 Affine Projection Algorithm (APA)

APA is a generalisation of NLMS that uses multiple past input vectors for the update. Instead of updating based on a single vector, it projects the error onto a subspace spanned by recent inputs.

Its update rule is defined as

$$w(n+1) = w(n) + \mu X(N)[X^T(n)X(n) + \delta I]^{-1}e(n) \tag{4.14}$$

Where:

- X(n) is the matrix of K past input vectors
- $e(n) = d(n) - X^T(n)w(n)$

APA is faster to converge compared to NLMS and more robust to correlated inputs but has a higher computational cost (O(MK)) and requires more memory.

4.2.1.6 Kalman filtering

Kalman filtering is a powerful recursive estimation algorithm used to estimate the state of a linear dynamic system in the presence of noise. It assumes both process noise and measurement noise are Gaussian and models the system using state-space equations.

Its system model consists of:

- a state transition (process) model:

$$x(n+1) = F(n)x(n) + w(n) \tag{4.15}$$

- and an observation (measurement) model:

$$z(n) = H(N)x(n) + v(n) \tag{4.16}$$

Where:

- x(n) is a hidden state vector at time n
- z(n) is the observed measurement vector
- F(n) is the state transition matrix
- H(n) is the observation matrix
- w(n), v(n) is the process and measurement noise (zero-mean, Gaussian)

Each iteration consists of two phases:

1) Prediction (a priori)
   a. Predict the next state:

$$\hat{x}^{-}(n) = F(n-1)\hat{x}(n-1) \tag{4.17}$$

   b. Predict the error covariance:

$$P^{-}(n) = F(n-1)P(n-1)F^{T}(n-1) + Q(n-1) \tag{4.18}$$

Where:

- Q is the process noise covariance

2) Update (a posteriori)
   a. Compute the Kalman gain:

$$K(n) = P^{-}(n)H^{T}(n)[H(n)P^{-}(n)H^{T}(n) + R(n)]^{-1} \tag{4.19}$$

Where:

- R is the measurement noise covariance

   b. Update the state estimate:

$$\hat{x}(n) = \hat{x}^{-}(n) + K(n)[z(n) - H(n)\hat{x}^{-}(n)] \tag{4.20}$$

   c. Update the error covariance:

$$P(n) = [I - K(n)H(n)]P^{-}(n) \tag{4.21}$$

Kalman filters performs optimally for linear-Gaussian systems and dynamically adapt to system changes. They do however require accurate models of noise and dynamics and is computationally heavy ($O(L^3)$), requiring multiple matrix multiplications and inversions, as well as being sensitive to parameter tuning.

4.2.1.7 Algorithm Considerations

Constant Modulus Algorithm (CMA) was considered in the filter design phase but didn't meet the requirements, given that it assumes the input signal has a constant modulus and attempting to force the output to have a constant magnitude, despite not needing a reference signal.

Adaptive Noise Cancelling (ANC) is a practical application of adaptive filtering. It uses a reference noise input and a primary signal (signal+ noise) to adaptively subtract noise. The filter learns to predict and cancel the noise component $e(n) = d(n) - \hat{n}(n)$ and often uses LMS/NLMS or RLS depending on performance requirements. It does however require a good noise reference, therefore was unsuitable for the proposed solution.

An additional idea of implementing the adaptive filters in the frequency domain was explored These would require additional "blocks" of frames that would need an FFT applied to them and additional memory expense. They would improve computational expense from the fact that convolution in the time domain becomes a multiplication in the frequency domain, as well as, from block-based processing. The implementation would rely on overlap-save/ overlap-add buffering [42]. This idea was not realised, given there was inadequate time to design and implement.

4.2.1.8 Selection for tasks

Given that the ESP32-C6 MCU is a RISC-V processor with limited floating-point performance through libraries (no dedicated FPU), the optimal adaptive filtering algorithm selected for implementation was the fast-converging RLS with a 14-point Tap Delay Line (TDL), since LMS/NLMS performed sub-optimally for correlated inputs and Kalman being very high in computational cost (not real-time).

RLS and Kalman were used in the ATF skin-characteristics' derivation process, described in section 4.2.3.

4.2.2 Spatial Filtering

Further investigation into other signal processing methods like the Hilbert Huang Transform (HHT), Trimmed Mean spectrogram, Wigner-Ville Distribution (WVD), Discrete Wavelet transform (DWT) and a simple Short Time Fourier Transform (STFT) [11], [15], as well as Principle Component Analysis (PCA) and Non-negative Matrix Factorisation (NMF) were planned, with the HHT, DWT, STFT being designed and tested together with the adaptive filters from section 4.2.1 but didn't provide additional spatial resolution.

4.2.3 Acoustic Transfer Function

In order to derive an Acoustic Transfer Function for the skin on the mitral point, an investigation was conducted using Electrocardiogram (ECG) and Phonocardiogram (PCG) signals and three distinct approaches: deconvolution via frequency domain division, adaptive filtering to get the channel response via Recursive Least Squares (RLS) and spectral estimation using MATLAB's *tfestimate()* function. The dataset used was from the PhysioNet electro-phono-cardiogram (EPHNOGRAM) database [43] and signal ECGPCG0003 was selected due to its low noise characteristics.

Both ECG and PCG signals were first downsampled to 1 kHz for computational efficiency. Bandpass filters were designed to isolate relevant frequencies in the regions of interest. To ensure time alignment, group delays from both filters were estimated using *grpdelay()* and the ECG signal was delay-compensated accordingly.

An RLS approach was implemented to estimate a linear model mapping the bandpassed PCG to the ECG signal. The RLS converged to a stable impulse response, which provides an estimate of the skin transfer function. The RLS approach outperformed a Least Mean Squared (LMS) algorithm developed in this context due to faster convergence and better tracking capabilities.

To explore linear convolutional relationships, a frequency domain division was implemented as follows:

$$h(t) = \mathrm{IFFT}\left(\frac{\mathrm{FFT}(\mathrm{ECG}(t))}{\mathrm{FFT}(\mathrm{PCG}(t)) + \delta}\right) \tag{4.22}$$

Where δ is a small number to avoid division by zero.

Finally, MATLAB's *tfestimate()* was used to compute cross- spectral density between the ECG and PCG and obtain a frequency domain estimate of the transfer function. This estimate was converted into a rational transfer function using *invfreqz()* for further analysis and visualisation.

All derived filters and frequency responses closely matched each other with an MSE of 22e-3 for the *tfestimate()* and the frequency domain division method and an MSE of 6e-3 between RLS and the *tfestimate()* – with proper coefficient truncation to match the size of RLS.

4.3 MICROCONTROLLER

4.3.1 Requirements

For the device to meet the functionality set out in this project, it needs to be capable of recording audio from a MEMS microphone, filtering the recording, writing data to non-volatile memory, and transferring the data over a Bluetooth Low Energy connection.

The device should have 3 operational modes, standby, continuous and low power. Due to time constraints the low power mode was left unimplemented.

4.3.2 Sound Data Gathering

To retrieve sound data from the MEMS microphone, 2 suitable approaches have been identified.

The first approach was to use an Inter-Integrated Circuit Sound (I2S) which is a serial protocol used for transfer of 2 channel digital audio. This approach works with preassembled MEMS boards that handle the analogue to digital conversion and transmit a digital signal to the microcontroller.

The second approach is capturing the analogue input of the MEMS via the onboard ADC (ADC_UNIT_1), configured on channel 6 with 12 dB attenuation. It was configured to 12-bit resolution, providing 4096 discrete quantisation levels:

$$\text{levels} = 2^{\text{bits}} = 2^{12} = 4096 \tag{4.23}$$

This imposes a quantisation noise floor and restricts the dynamic range of the input signal, effectively shaping how the analogue waveform is digitised before entering the DSP pipeline. Any noise or distortion introduced at this stage propagates through the filter chain, reinforcing the importance of precise calibration and filtering. This also occurs in the I2S method, but is all introduced internally.

Due to non-linearities and variations across the ESP32 silicon, curve-fitting calibration was employed using Espressif's *adc_cali_raw_to_voltage()* function. This translates raw ADC values into millivolt readings. A fallback path using uncalibrated offset values ensured functionality on chips without eFuse calibration data, preserving deployment robustness. The value for DC calibration was measured at 1.387V, closely matching that mentioned in the datasheet (1.35 V).

The internal ADC was selected for the final deployment of gathering data from the microphone as it allowed for any type of mems microphone to be used. If the I2S approach was used the microphone selection would have been limited to solutions on the market which implement the I2S protocol, greatly limiting the selection.

The microcontroller samples the MEMS connection at rate of 1 kHz for 10 seconds at a time which forms the raw data of the sound file.

4.3.3 Data Filtering

To support signal conditioning and adaptive modelling in real-time, an embedded processing pipeline was implemented directly on the ESP32C6 MCU, leveraging the ESP-DSP library for optimised digital signal processing.

The input signal was first passed through a five-stage bi-quad filter cascade, implemented using second-order sections (SOS) defined by carefully designed Infinite Impulse Response (IIR) filter coefficients in MATLAB's *FilterDesigner* tool. Each SOS was applied in sequence using the *dsps_biquad_f32*() function, with per-stage state maintained in a matrix to preserve continuity across incoming samples. Each section in the SOS matrix was defined by five floating-point coefficients of the form:

$$\{b_0, b_1, b_2, a_1, a_2\} \tag{4.24}$$

Where the leading denominator term a_0 is assumed to be 1.0, with this omission being standard in Direct Form II bi-quad implementations, including in the ESP-DSP library's *dsps_biquad_f32*() function, which simplifies the coefficient structure and reduces memory overhead. The absence of a_0 necessitates prior normalization of all other coefficients, ensuring numerical stability and correct gain scaling.

Each SOS stage corresponds to a second-order IIR filter, and with five sections total (excluding a_0), the overall filter achieves a 10^{th}-order response. The filter structure allows for fine-grained control over the frequency response while maintaining numerical stability. This is especially important in embedded systems with limited precision. The coefficients themselves were exported from MATLAB into a .c header file and stored as 32-bit floating-point numbers in .8f precision format. This represents a balance between numerical clarity and storage readability during debugging and code maintenance. This level of precision is typically sufficient for most embedded audio applications, where the input signal's resolution is already limited by the ADC. The SOS filter's coefficients were carefully designed to mitigate out-of-band noise and shape the signal spectrum within the limits imposed by the ADC quantisation. The details of implementation for the SOS filter, can be seen in Figure 5 through Figure 7.

This IIR filtering was used to isolate and amplify frequency bands of interest while minimizing memory overhead – each bi-quad operated in real-time on a per-sample basis, eliminating the need for large intermediate buffers.

Following this the signal was further processed via a 64-tap Finite Impulse Response (FIR) inverse filter to remove acoustic attenuation through skin tissue. This was implemented via convolution with a set of precomputed coefficients (derived in section 4.2.3) and used a circular buffer to maintain the memory efficiency. The convolution operation effectively shaped the signal to mimic how it would be before passing through biological tissue, forming a "desired" signal that was later used as a reference in adaptive modelling.

A core component of the implementation was a Recursive Least Squares (RLS) adaptive filter with a tap delay line length of 14. This structure was also maintained in a circular buffer of past filtered values and updated a set of weights in real time to minimise the error between the predicted and desired signals. The Kalman gain, inverse correlation matrix S and weight updates were all computed at each iteration. Fast vector operations, including dot products used in both prediction and update phases were offloaded to the ESP-DSP library via *dsps_dotprod_f32()* for computational efficiency on the RISC-V core. A variable gain control system was also implemented, adjusting the signal amplification factor (gain_factor) in response to the signal envelope to avoid clipping and maintain dynamic range.

Memory allocation was split consciously between heap and stack. Fixed-size buffers used for sample processing, such as the circular buffer for convolution, the inverse correlation matrix S and the SOS state matrix were allocated statically on the stack. This allowed for deterministic timing and eliminated heap fragmentation risks during real-time execution. File output, including the WAV headers and sample array, was written to the heap via buffered I/O. This hybrid memory model ensured that real-time operations could run uninterrupted on the stack, while larger, non-critical operations such as file writing functions made efficient use of heap resources.

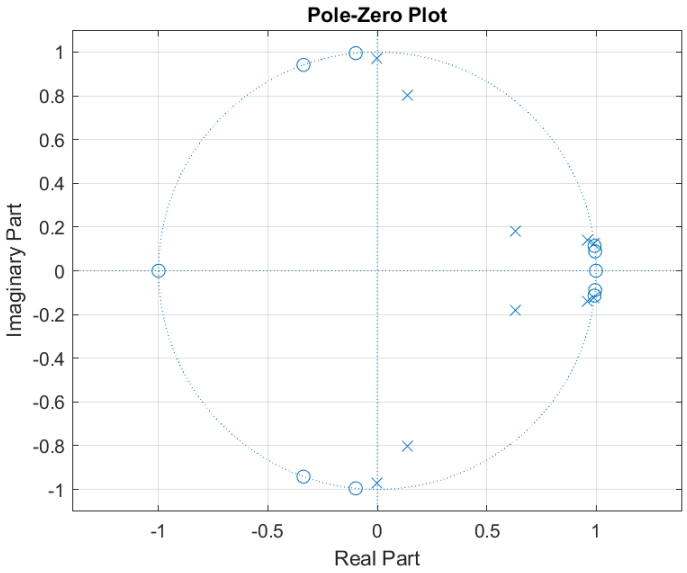


Figure 5. Pole-zero plot for SOS matrix.

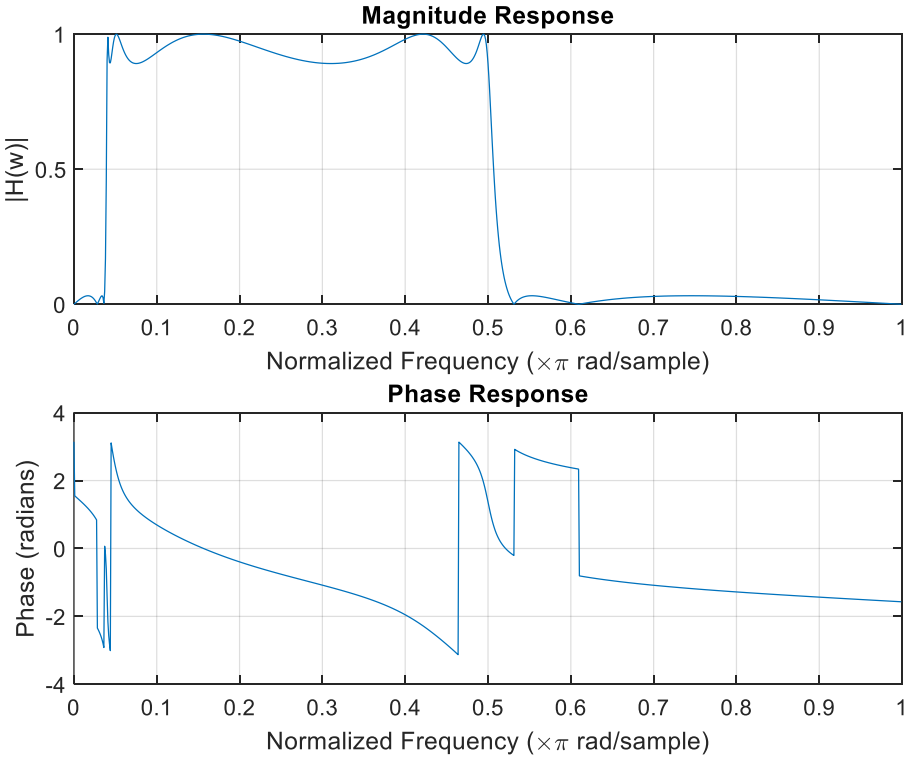


Figure 6. Magnitude/ phase response for the SOS matrix.

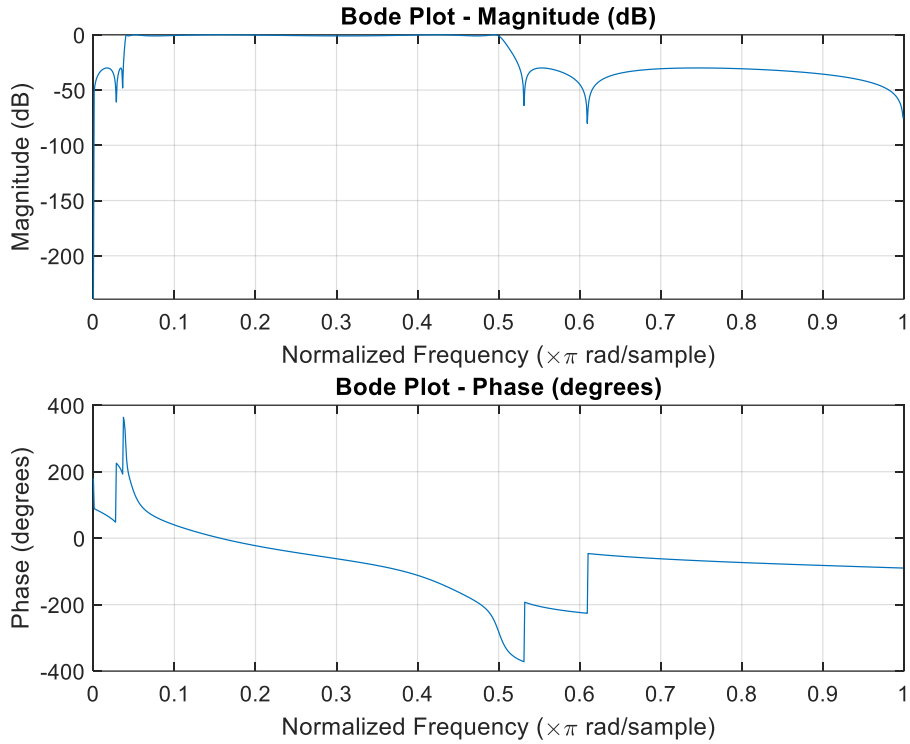


Figure 7. Bode plot for magnitude/ phase response of the SOS matrix.

4.3.4 Data Storage

The ESP32-C6 model used in this project has 8 MB of internal flash memory which can be configured into partitions to carry out different functions. For storing and reading the recordings as files, the memory needs to be partitioned as file system.

To implement the file system partition in the NOR memory of the ESP32, Serial Peripheral Interface Flash File Storage (SPIFFS) was used. SPIFFS is easy to set up, making it a great choice for the project. SPIFFS doesn't allow for folder structures inside of the file system partition unlike to other systems such as LittleFS. However, this functionality will not be needed as in this case all files would be uploaded to the same folder. If the device was to store more than one type of data, other systems would be more suitable.

LittleFS is known to have significantly slower read and write speeds compare to using SPIFFS which would be an issue for this application as it requires fast read and write speeds.

A storage partition was set to the largest value possible, taking up the rest of the free space on the microcontroller at a size of 7274496 bytes (0x6f0000). To access data within this partition the prefix "/stotage" is used to find the intended data.

The audio recordings are stored as .wav files with their name being a timestamp of when they were recorded in "YYYY-MM-DD-HH-MM-SS.wav" format.

4.3.5 BLE Implementation

The project used the ESP32 BLE sample code as a guide and a starting point on how the Bluetooth communication should be implemented on the ESP32 platform [44].

Bluetooth Low Energy was used for short range communication and data transfer between the MCU and android application. It operates on a server-client system where the MCU acts as a server which advertises/broadcasts its presence through Generic Access Profile (GAP). This allows any nearby

devices to connect to the server and subscribe to the advertised services using Generic Attribute Profile (GATT) which establishes how data will be organised and exchanged. Once a client link is established and the client has subscribed to some service the transfer of data from server (MCU) to client (mobile application) can begin.

The BLE on ESP32 can generally be implemented using one of 2 libraries; Bluedroid or NimBLE. Bluedroid is full stack Bluetooth solution meaning it includes both Bluetooth Classic and Bluetooth Low Energy. NimBLE is an open-source solution created as a direct competitor/successor to Bluedroid with nearly 50% reduction in flash usage and about 100 kb less RAM usage. Unlike Bluedroid, NimBLE implements only the Bluetooth Low Energy functionality [45]. For this project NimBLE was a clear choice as the wireless device will be communicating solely over BLE and allows us to keep the memory usage to a minimum.

4.3.5.1 Generic Access Profile (GAP)

GAP is a foundational BLE protocol that defines how devices discover, connect, and interact with each other. A device can take one or more of 4 operating roles: peripheral, central, broadcaster or observer:

- Peripheral (server) advertises its presence for other devices to connect to (e.g. smart watch, health monitor, etc.)
- Central (client) scans for nearby peripherals that it can connect to and initiate a connection with
- Broadcaster sends out non-connectable advertisements containing data (acting as a beacon)
- Observer scans for device advertisements but doesn't connect (used for data gathering)

In this project the device acts as a peripheral which advertises its existence to establish a connection. The mobile phone (with accompanying application) then acts as the central device which knows how to interact with the peripheral.

Once a connection with the central device has been established, the peripheral device stops advertising and only shares information with its paired device.

4.3.5.2 Generic Attribute Profile (GATT)

The GATT defines how data is organized, formatted, and exchanged between BLE devices once a connection is established. It operates on top of an Attribute Protocol (ATT) and defines data interactions such as reading, writing, notifying, and indicating characteristics.

Characteristics are contained within GATT services which are advertised to the client along with their description. It is up to the client to then indicate to the peripheral which services and characteristics they would like to read information from or write information to. There are two methods a client can use to get information about a characteristic, Notify and Indicate.

To be notified about a characteristic a client must subscribe to it and leave their communication line open for any incoming data. The peripheral is then able to send any new data it gathers directly to the central device without needing to directly read the data from the peripheral.

Indicate works very similar to notify where the client must subscribe to a characteristic first, however the central device must then respond with an acknowledgement after each message from the peripheral. This gives the peripheral an assurance that the transferred data has been correctly received and wasn't lost or corrupted in transmission. Using indication over notify takes significantly longer to transfer data as it is two-way communication rather than just one-way.

Each service and characteristic is assigned an Unique Universal Identifier (UUID) which can be either a 16-bit identifier part of the standard Bluetooth list or a 128-bit identifier used for custom data formatting and communications.

The device has implemented a standard Object Transfer Service (OTS) with an object data characteristic which was used for the data transfer between the peripheral device and the central. The characteristic was set to be an indication characteristic as it allows the peripheral to send new data whenever it becomes available.

The device has also implemented two custom services, each with a unique characteristic. The first unique service is a time synchronisation service. The microcontroller has an internal clock which starts counting when the device is powered on however it has no way of relating the timer to actual real-world time. The service includes a read/write characteristic for setting the peripheral device's time to the current UTC time. This was very important as it allows for each recording to be given its unique timestamp.

The second unique service used on the peripheral device is a device mode setting service. This was used to set the operational mode of the device. The service also has a read/write characteristic where the central device would be able to change the mode. The device has 3 modes of operation for the client to switch between: standby, low power and continuous. As stated previously the low power mode has not been implemented and is just a placeholder.

4.3.6 Data transfer

To transfer data between the device (server) and the application (client) an indication Object Transfer Service (OTS) was used. When new data is available to be sent and a connection is present, the storage partition is scanned for any files currently on the device. These are then sent over in turn and then deleted form the peripheral device.

BLE has a Maximum Transmission Unit (MTU) of 256 bytes. Each transmitted packet has a header of 3 bytes leaving 253 bytes for data. The file transfer begins by loading in a file. The first packet that is sent to the central device contains the name of the file being transmitted. The rest of the packets are the file payload. The transmission is ended with the transmission of a single data packet containing the number one to indicate to the receiver that the entirety of the file has been transferred. Since the indication characteristic is being used, a semaphore system allows a maximum of 4 packets to be sent concurrently before the microcontroller will wait to receive an acknowledgement and send another packet. The data is sent in the order in which it appears in the file.

In the case that an acknowledgement is not received within 5 seconds of sending the data packet, the file transmission will come to a halt and restart from the very beginning by sending the file name again. The chances of the acknowledgements not arriving are very slim and this has occurred in less than 1% of the tests carried out thought out this project.

While this approach can be power and data inefficient (resending a whole file for one dropped/corrupted packet), it is a minor drawback compared to implementing a large, ordered packet structure on the microcontroller. This would require keeping a copy of last N transmitted packets and a second structure in the application for stitching the file's data back together on the other side.

4.3.7 Task Scheduling

The microcontroller has 3 operating modes standby, continuous and low power mode:

Standby: there are no tasks bring carried out, but the Bluetooth remains active to detect connection or mode changes from the central device.

Continuous: the device initially records 10s of audio from the microphone, filters the data and saves it. It then calls a task to that send over data to the application. These two processes call each other in a loop until there is a change in mode from the central device.

Low Power: the intention (while unimplemented) would be to carry out the same function as continuous except it would record 1 minute of data (split into 10 second segments) every 10 minutes. It would then send these over as a block of data. While the device is not sampling or sending it should enter into a mode similar to standby which only runs the Bluetooth as a background task to detect changes in mode or connection status. This should improve the battery life of the device for long-term wear while still recording often enough to capture any arrythmias.

To schedule the tasks correctly and effectively on the microcontroller's single core CPU, FreeRTOS was used. It is an open-source real-time operating system (RTOS) kernel that is natively implemented by the ESP32 microcontroller. Using FreeRTOS allows the microcontroller to concurrently carry its tasks set out in the main functionality loop as well as run the Bluetooth connection without any issues.

The Bluetooth tasks in this project were set a higher priority than any of the core functionalities as any incoming Bluetooth messages should be handled immediately as to not cause a communication timeout. These messages could also contain information about a change in the operating mode of the device which should be handled and updated immediately as it will have a direct effect on the running of scheduled tasks.

A *vTaskDelay()* function, part of the FreeRTOS suite, was used to create a short wait period in non-critical parts of the code as well as for the time delay between sampling the microphone. In its standard mode vTaskDelay can deliver a delay as small as 1 millisecond which gives 1 kHz sampling rate. Going any smaller with wait times using vTaskDelay can result in the process being out of time by up to 1 tick – doubling the expected wait time. An ESP32 delay function was also considered for the sampling of the microcontroller "*esp_rom_delay_us()*". This is capable of creating an accurate delay on the microcontroller as small as 1 microsecond. The main issue with using esp_rom_delay_us() over vTaskDelay() is that esp_rom_delay_us() does not free the CPU for other processes to run while it is waiting. This violates FreeRTOS protocol which stops any function from holding the CPU thread for longer than 5 seconds at a time before allowing other processes to use it.

4.4 PRINTED CIRCUIT BOARD (PCB)

4.4.1 Circuit Schematic

The circuit design is shown in Figure 8. It was split into three main sections: power, sound, and the ESP32-C6 microcontroller.

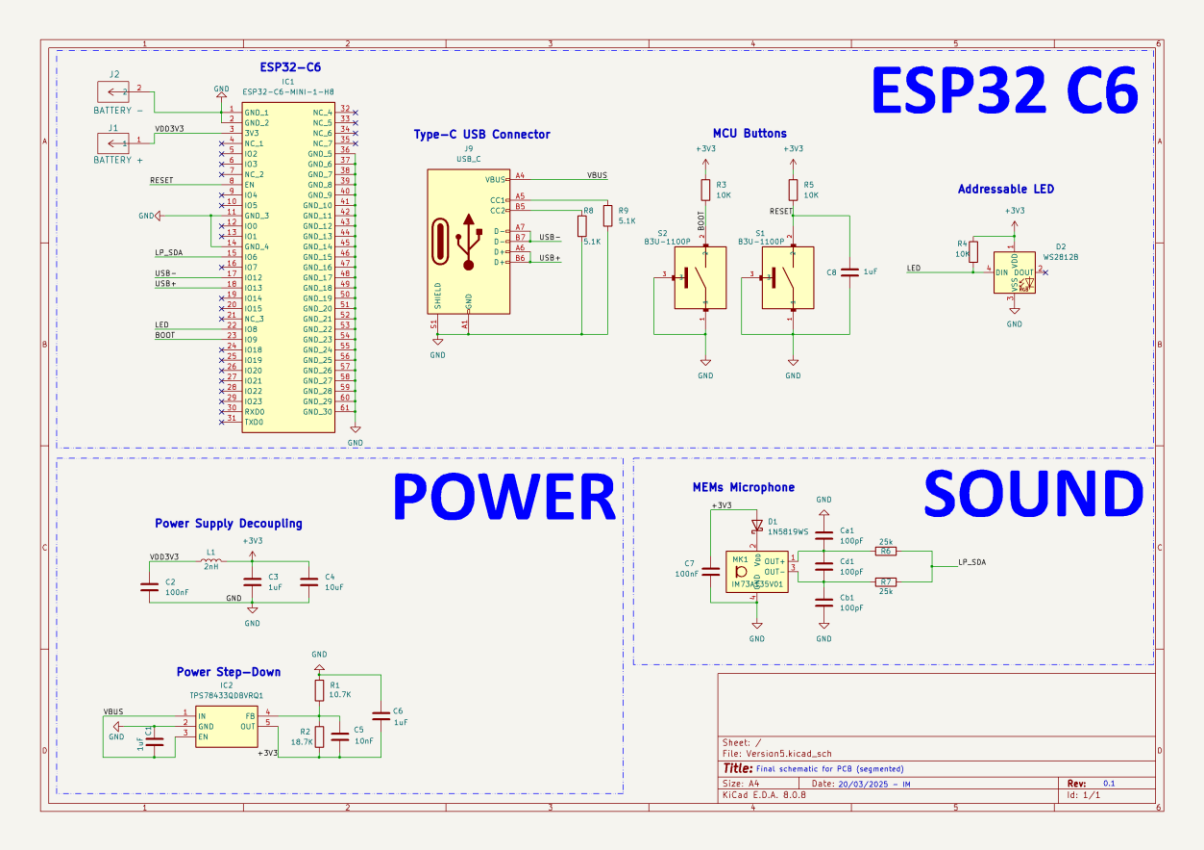


Figure 8: Circuit schematic (segmented view)

The power section included a low-dropout voltage regulator (LDO) responsible for stepping the 5V USB input (or the 3.7V battery input) down to the 3.3V required by the rest of the circuit, as well as a small decoupling circuit to balance the supply with the ESP32's variable internal resistance and stabilise any fluctuations. The sound section was centred around the IM73A135V01 MEMs microphone, with the surrounding circuitry following the recommended datasheet layout [22]. The MEMs output (LP_SDA) was tied to an input line of the microcontroller.

The microcontroller section included two buttons on the 'BOOT' and 'REBOOT' lines, and a Type-C USB connection to allow for the upload of code from a laptop. The 3.3V power and ground lines were tied to single-pin female headers (J1, J2) for the battery connections to ensure the battery could be fully detachable from the circuit. With this configuration, the system could be powered either through the USB connection for modifying/debugging of code, or through the battery for full system verification. This section also included an addressable LED (i.e. an LED from which colour can be modified based on code output from the microcontroller). This was used during testing to verify when the system was powered, paired in Bluetooth, and sampling. The LED addressing was removed from the final version of the MCU code.

For a clearer understanding of each section's purpose in the system, Figure 8 displays them separately. A full view of the circuit with all modules combined can be seen in Appendix E: Circuit Schematics.

4.4.2 PCB Design Requirements

4.4.2.1 Components & Footprints

The PCB was designed using KiCad EDA software, chosen for its flexibility, backup scheme, and support for importing external component libraries. While most circuit components were available in standard KiCad libraries, some required sourcing from external libraries. Notably, a custom footprint for the ESP32-C6FH8 microcontroller was requested from and provided by Mouser Electronics. In later stages of the project, component selections had to be updated to reflect available stock levels. Detail on the purpose and reason for each of the core components in the PCB design can be seen below in Table 7.

Table 7. PCB component selection and justification

Component	Footprint	Circuit Module	Justification
ESP32-C6H8 Mini Chip	Mouser Electronics	ESP32-C6	Best MCU selected in component testing
USB2.0 Connector Type C	Digikey UK	ESP32-C6	On-board connecter for flashing MCU code
Switch Tactile SPST-NO	Snap EDA	ESP32-C6	BOOT and REBOOT buttons for flashing MCU code
Smart LED WS2812B Pixel	Digikey UK	ESP32-C6	Addressable for testing
LDO Voltage Regulator	Mouser Electronics	Power	When USB connected, drops 5V supply down to 3.3V
MEMs Microphone	Digikey UK	Sound	Best microphone selected in component testing
Schottky Diode	Digikey UK	Sound	Drops the 3.3V source down to 3.0V for the microphone
1x1 Female Socket	Snap EDA	ESP32-C6	Connectors for fully detachable battery
Resistors (5.1k, 10k, 10.7k, 18.7k)	KiCad EDA	ESP32-C6, Power, Sound	All 0805 SMD – smallest size to reliably hand-solder within group capabilities
Resistors (25k)	KiCad EDA	Sound	0603 SMD due to supplier stock levels
Capacitors (1u, 10u, 10n, 200n, 100p)	KiCad EDA	ESP32-C6, Power, Sound	All 0805 SMD – smallest size to reliably hand-solder within group capabilities
Inductor (2n)	KiCad EDA	Power	0402 SMD due to supplier stock levels

A Bill of Materials (BoM) for the final PCB design is included in Appendix F: PCB Bill of Materials. A Bill of Materials for the project as a whole and for every iteration of the PCB prototypes is included in Appendix H: Complete Bill of Materials and discussed further in Section 10.

4.4.2.2 Layout

The purpose of the PCB was to bring together all hardware elements to form one device that could reasonably and comfortably fit inside the enclosure/brace. Given the importance of maintaining clean signals and reducing all sources of noise – including electrical – the priority in design was to reduce size and keep core components as close together as possible to minimise track inductance. Reducing the size would also help to minimise weight, cost, and complexity of the enclosure design – all of which were particularly important for a wearable device.

4.4.2.3 Power

To keep the device as small as possible, the aim was to power it from two coin-cell (CR2032) batteries, which each typically deliver between 3.0 and 3.3V. These were to be connected in parallel to increase the total current supply. Rather than increasing the width or breadth of the PCB to accommodate the batteries, two additional PCBs containing only the battery enclosures were designed to stack onto the first 'base' PCB using header pins. During integration testing, the coin-cell batteries were replaced with a single rechargeable 3.7V battery that connected to the base header pins, and instead of the stackable PCBs, the battery enclosure was incorporated into the device enclosure to keep the size to a minimum (discussed further in Section 7.2: Enclosure).

4.4.2.4 Manufacture

One of the main restrictions in the PCB design was the manufacturing process. If it was to be manufactured internally (i.e. within the university), then the cost would be lower and lead times shorter, but all designs would need to fit the university limitations, and all components would need to be hand-soldered. If it were to be manufactured externally through a university-approved supplier (i.e. JLC), lead times and cost would be considerably greater, but the design could be more specialised, and all components could be machined onto the board. It was decided to use internal manufacture for all prototypes and then, if time allowed, use external for the finalised design. Following the university manufacture guidelines (concerning minimum track widths, clearances, thermal reliefs, etc), the PCB layout went through several design iterations to find the best balance between project requirements and manufacturing capability.

4.4.3 Design Iterations

4.4.3.1 Version 1 – Manufactured Prototype

Initial layout designs were based on the university manufacture guidelines and reviewed as the circuit schematic went through updates. The first manufactured iteration of the PCB layout is shown below in Figure 9. Some of the tracks had to be less than the recommended 1.5 mm to fit the footprints correctly (0.4 mm for MCU, 0.2 mm for USB); the clearances then also had to be reduced when these tracks ran alongside each other. Drill holes were also reduced to the absolute minimum of 0.6 mm where necessary for the USB and MCU tracks. Power planes were used to reduce inductance between USB and MCU. The overall size was 59 x 54 mm.

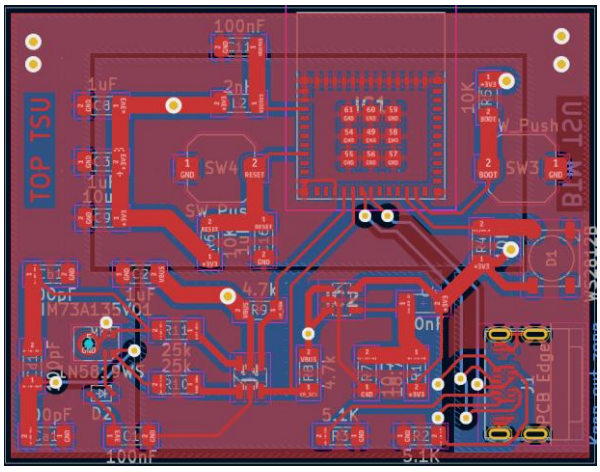


Figure 9: PCB Version 1 - First manufactured iteration

Version 1 had a lead time of 5 days, after which all components were hand-soldered. The MCU and MEMs microphone were bottom-terminated components (BTC), so were placed onto the board using

solder paste and a heat gun. During continuity checks and board validation, several issues with the manufacture became apparent. Narrow clearances around the vias meant that the copper planes sometimes 'leaked' over and bridged with the vias, the 0.2 mm and 0.4 mm traces were more liable to being removed or damaged during soldering, and in the absence of copper-plated vias, soldering a wire through without leakage was a challenge.

To get around this, adjustments were made to Version 1 to use it as an initial prototype – useful while the schematic also underwent changes (such as removing the external ADC). Unnecessary components were bridged with zero-ohm resistors, vias were connected to nearby tracks, and copper leakage was filed off.

4.4.3.2 Version 2 – Development Iteration

Between Version 1 and Version 2 (Figure 10), the layout was adapted and reviewed continuously to enable smooth integration with the rest of the project elements. For example, after deciding on the mitral auscultation point following tests with the machine learning model, the MEMs microphone was moved to as close to the edge of the PCB as possible. Following development with the brace design, this needed to be the shortest (y) edge, with the PCB to be extended along the x-axis instead. Two additional power PCBs were designed to hold the batteries, with all three to be stacked using header pins.

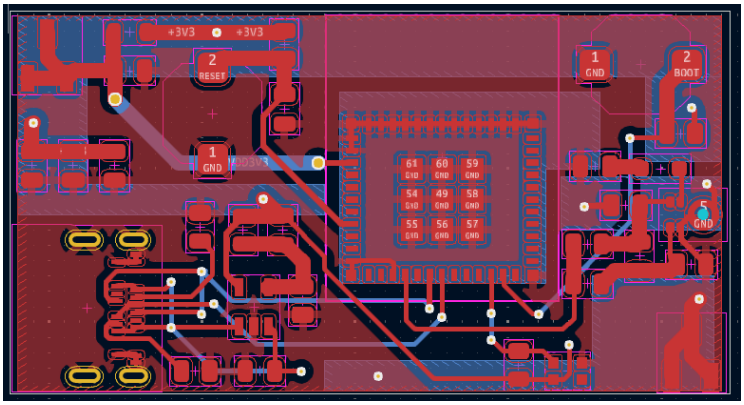


Figure 10: PCB Version 2 - Development iteration

The size was 47 x 27 mm, as this was the smallest it could be while allocating enough space for the battery enclosures (35 x 26 mm). Footprints were updated to reflect changes to the schematic, and maximum SMD size was reduced from 1206 to 0805 to save more space.

4.4.3.3 Version 3 – Manufactured Prototype

Version 2 was not manufactured; it instead acted as the basis for the final PCB layout (Figure 11). Changes included updating choice of peripheral components (buttons, resistors, capacitors, etc.) to match with supplier stock levels/availability, cutting out space beneath the USB-C connector for easier mounting, and including 4 mounting holes in each corner to ensure the PCB fit neatly into the printed enclosure. The ADC was moved closer to the MEMs to reduce track inductance/noise. The VBUS power plane was replaced with a 3V3 power plane. 1 mm tracks were used for all but the MCU and USB pads. Most importantly, to avoid the issues with via leakage and soldering present in Version 1, all vias were given a 1 mm clearance to any copper and spaced out as much as possible within the design.

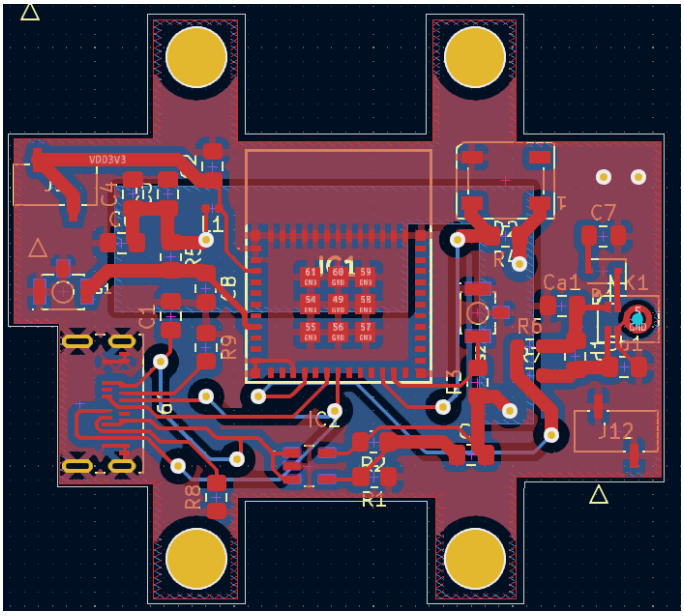


Figure 11: PCB Version 3 – Second manufactured iteration

Version 3 (including the 2 battery PCBs) had a lead time of 2 days, and all components (except the BTCs) were hand-soldered. With the updated design, there was no leakage or via errors during manufacture. Vias were soldered through using 0.6 mm wires and were given enough clearance to make it possible to hand-solder. During continuity checks and board validation for Version 3, no errors or shorts were discovered. The battery PCBs stacked onto the base PCB using headers, giving a size of 42.5 x 47 x 34 mm.

4.4.3.4 Version 4 – Final Iteration

Small adjustments to the Version 3 layout were made during integration testing; this became the final PCB design shown in Figure 12. Space was cut out below the antenna of the MCU, as the previous copper plane running below it caused too much interference for Bluetooth to successfully connect. With the change of the coin-cell batteries for a single 3.7V battery, the 1x2 pin sockets were replaced with 1x1 sockets that connected to a system on-off switch embedded into the device enclosure.

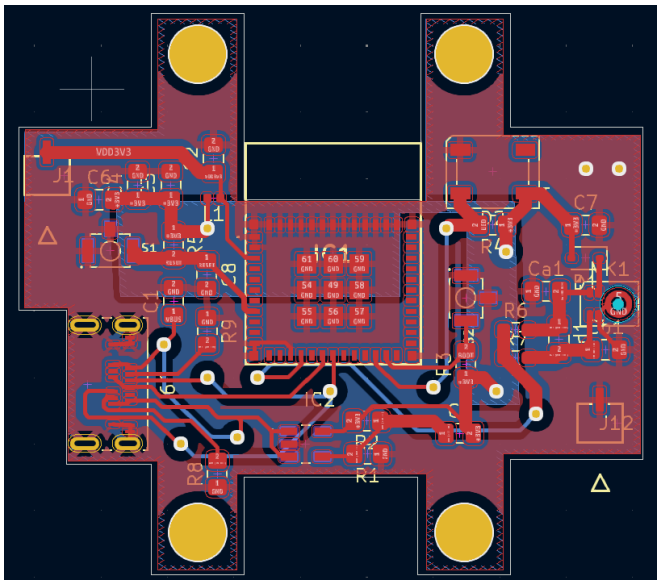


Figure 12: PCB Version 4 - Final iteration

With the battery excluded, the final size of the PCB was 42.5 x 47mm. It had ground planes on the top and bottom, as well as a 3.3V plane on the bottom. No errors or shorts were discovered in continuity checks, and there was sufficient inductance reduction for the MEMs to be able to record a signal, send it to the MCU, and for the MCU to transmit it to the application via Bluetooth within a reasonable time.

Detailed images of the final layout, including plane views of the top and bottom, are available in Appendix G: PCB Layout Schematics.

4.4.4 Manufacturing Process

Both manufactured iterations - Version 1 (Figure 9) and Version 2 (Figure 11) – were manufactured within the university. Lead times were between 2 and 5 working days. BTCs were soldered using a heat-gun and solder paste with a low melting point, following soldering profiles provided by the component manufacturers. All other components were hand-soldered in the university MEng Lab.

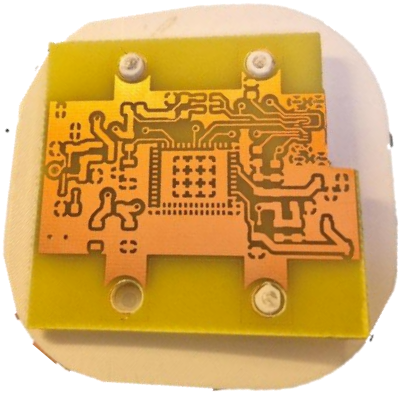


Figure 13: Version 4 (unsoldered) in early iteration of enclosure backplate

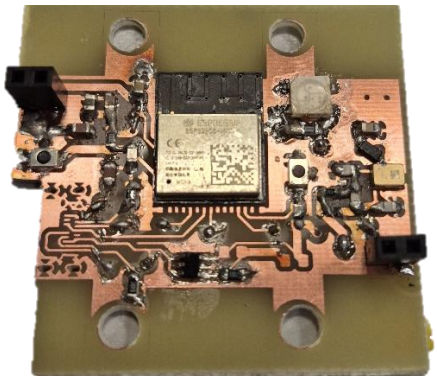


Figure 14: Version 3 (soldered)

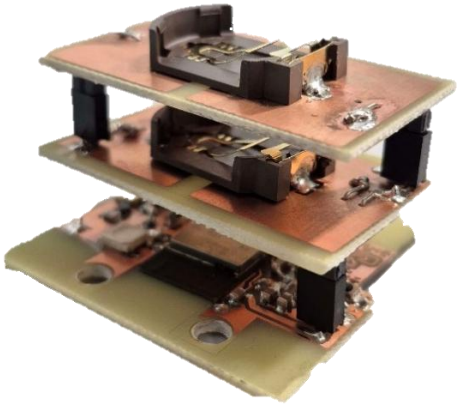


Figure 15: Version 3 (soldered) with additional battery stack PCBs

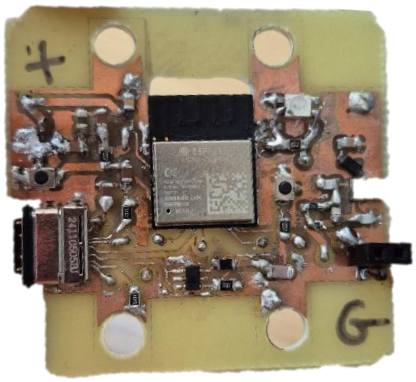


Figure 16: Final manufactured PCB

To keep the PCB secure against the users' chest, it needed to fit tightly into the enclosure. Later PCB design iterations (starting after Version 2 and the introduction of the four mounting holes) were developed in tandem with the enclosure to ensure compatibility. More detail on the enclosure can be found in Section 7.2: Enclosure.

5 APPLICATION

5.1 REQUIREMENTS ANALYSIS

The first stage of application development was defining the Functional and Non-Functional requirements. This was done with the application functionality and the end-user in mind. The requirements have been prioritised according to the MoSCoW structure (Must, Should, Could, Would) [46] and are shown in Table 8 and Table 9.

A few of the requirements changed between initial definition and final deliverables (as is common for professional, long-term projects). These are indicated with a * and a note to why they changed at the end of each table.

Table 8. List of functional requirements

Req. No.	Req. Description	Req. Priority
1.0	The system shall display instructions on fitting the device.*	Should
1.1	The system shall test if device is positioned correctly.	Could
1.2	The system shall use visuals to accompany instructions.	Should
2.0	The system shall display a list of recordings.	Must
2.1	The system shall sort recordings by date/time recorded.	Must
3.0	The system shall search/filter displayed recordings:	Should
3.1	By date/time recorded.	Should
3.2	By arrythmia detected.	Should
3.3	By user tag.	Could
4.0	The system shall playback recordings:	Must
4.1	Using audio.	Must
4.2	Visually.	Should
5.0	The system shall allow users to tag recordings:	Could
5.1	With pre-existing tags.	Could
5.2	With user defined tags.	Would
6.0	The system shall allow user to download recordings:*	Must
6.1	Individual recordings.	Must
6.2	Groups of recordings sorted by a criterion.	Could
7.0	The system shall allow user to delete recordings:	Must
7.1	Individual recordings.	Must
7.2	Groups of recordings sorted by a criterion.	Could
7.3	All recordings.	Must
8.0	The system shall allow users to input personal/health data to affect their ‘diagnosis’.*	Could
8.1	The system shall allow users to delete any/all personal info.	Must
9.0	The system shall confirm whether user is detecting or monitoring a condition and alert accordingly.	Would
10.0	The system shall display a list of all recent alerts (including arrythmia, suggestion and a link to the recording).*	Should
11.0	The system shall send a push notification when an arrythmia occurs.	Would
12.0	The system shall facilitate Bluetooth setup of the device.	Must
13.0	The system shall display device info (e.g. battery level).	Should
14.0	The system shall facilitate device management.	Must
14.1	Change device mode (Low power, Spaced recording, Constant recording).	Should
14.2	Sync data from device.	Must
15.0	The system shall provide option of application password protection.	Would
16.0	The system shall encrypt stored data.	Could

*Changes:
Requirements 1 and 10: while this stage will be vital for a final product, it was deemed more important to focus on functional aspects of the app for this time scope.
Requirement 6: since the recordings are stored on their device there is no need to 'download' them. However, instructions are provided on how to access them/save to another device.
Requirement 8: specific tracking of health conditions was found to be out of scope, relative to both resources and expertise. Therefore, no personal data is logged so no personal data needs to be deleted.

Table 9. List of non-functional requirements

Req. No.	Req. Description	Req. Priority
17.0	The instructions should be simple to follow and step-by-step.*	Must
18.0	The core functionality should be understandable by users unfamiliar with tech	Must
19.0	There should be accessibility settings.	Would

*Changes:
Requirement 17: this requirement is irrelevant as the instructions/fitting were deemed out of scope.

5.2 VIEW DESIGN

The next stage was designing the Graphical User Interfaces (GUIs) based on the above requirements. The design consists of 5 key screens (Figure 17):

- *Home Page*: links to all other pages, including Settings and Notifications
- *Notifications*: displays a list of all recent alerts which each include the arrythmia detected, the severity level and advice, and a link to the recording
- *Settings*: will let users update important preferences and information
- *Recordings*: displays a list of recordings that can be sorted, filtered and searched; when clicked, a recording can be played or visualised
- *Device Settings*: sets up Bluetooth connection to the wearable device, displays key info about the device, and manages certain device settings

Each of the designs is as simple and user friendly as possible, considering that end users are likely to be elderly and/or disabled.

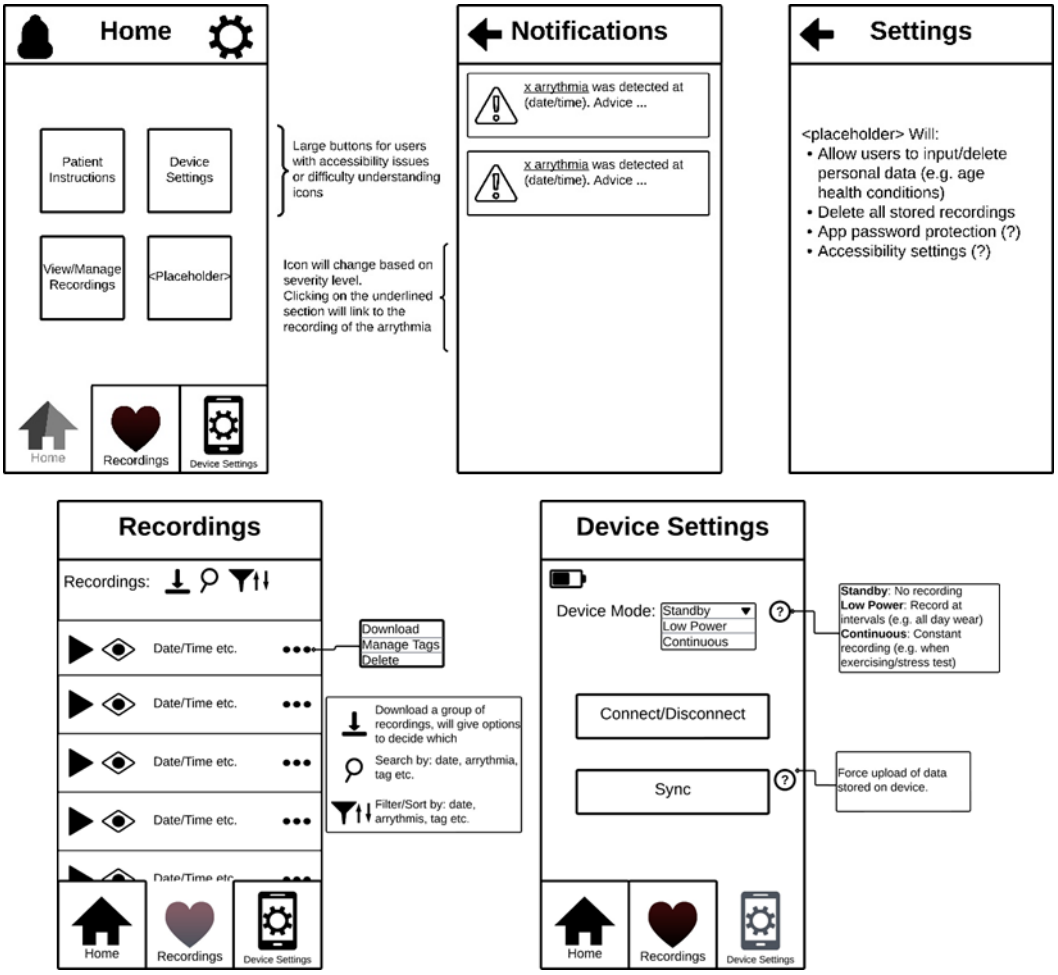


Figure 17. Initial GUI design

5.3 BACKEND DESIGN

An initial (basic) class relationship diagram was also created to identify the expected classes and their relationships to each other, as well as aid in discussions around the application backend. This loosely follows the Model-View-Controller structure, where the Model represents most of the computation and data handling, the View handles displaying to the user, and the Controller links the two. This initial class diagram is shown in Appendix A.

The finalised design that mirrors the end implementation is shown in Section 5.5. It should be noted that due to the nature of mobile app development (where the Views and their function are much more closely linked than that in general Software programming) the Model-View-Controller structure was not kept.

The application was also programmed using the Object-Oriented programming style which focuses on each class representing an ‘Object’. Where an Object is something that holds data relevant to itself and performs operations on that data. These classes can then interact to perform larger operations. The data that each class holds, and the other classes they know about should be strictly monitored to ensure it is relevant.

5.4 VIEW IMPLEMENTATION

The view implementation consists of a series of xml files that describe the components on the screen and provide IDs for the backend code to reference and add interaction to. These are:

- activity_home_page.xml
- fragment_device_settings.xml
- fragment_home.xml
- fragment_recordings.xml
- manage_tags_view.xml
- recording_item_loading.xml
- recording_view_item.xml
- tag_app_added.xml
- tag_user_added.xml

Each of the corresponding views can be seen in Appendix B. It should be noted that these aren't exactly what is displayed as some aspects are set programmatically and other things are only shown on dynamic interaction.

5.5 BACKEND IMPLEMENTATION

The application development focused on the three key screens: Home, Recordings, and Device Settings. As discussed in Section 5.1, the device fitting instructions and notification screen were out of scope. This was also decided for the user settings as the focus was on getting a polished final deliverable, rather than on user customisability.

5.5.1 Backend Classes + Functionality

A condensed version of the final class diagram is shown below (Figure 18), however a full version with fields and methods can be found at Appendix A.

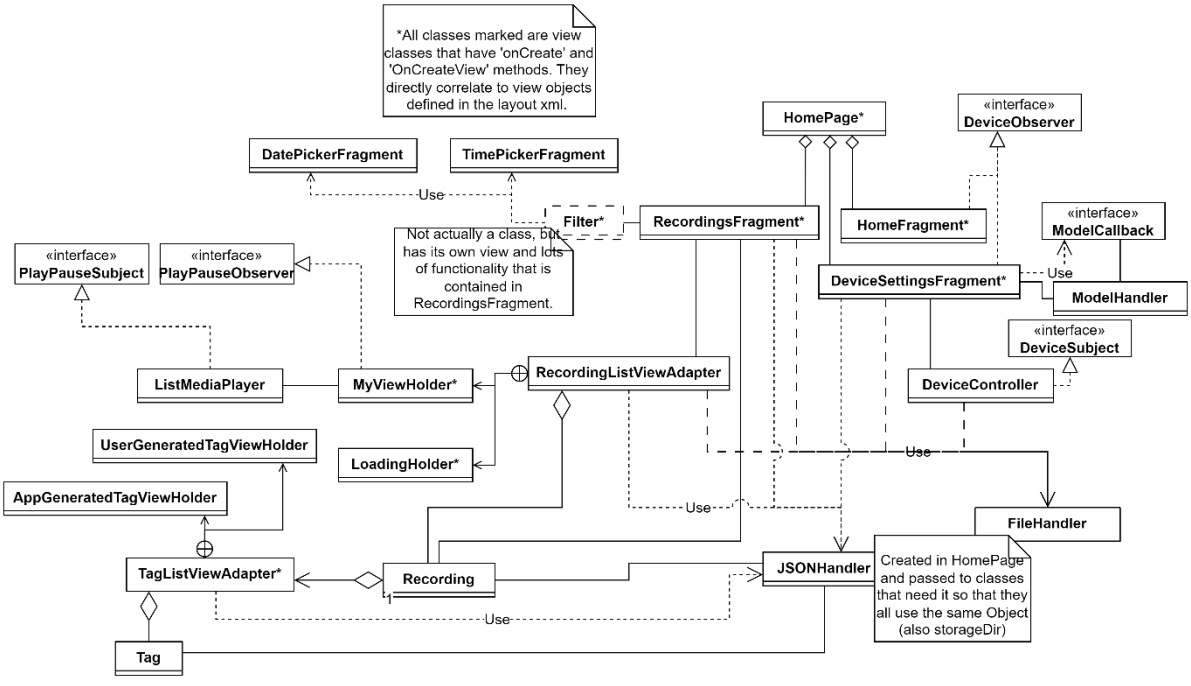


Figure 18. Condensed UML class diagram

Each of the starred classes is one that directly represents one of the views described in section 5.4. The rest of the classes are used by these view classes to perform processing of some kind. The rest of this section provides an exhaustive description of the purpose of each class.

*5.5.1.1 Home Page**

This is the launcher class which means it is the first view shown and code run when the app is opened. It is responsible for setting up the bottom navigation bar and instantiating each of the Fragments which hold the content to be switched between. It also sets up the toolbar which is displayed at the top of the screen showing the page title and any page buttons. It also instantiates the background threads for the Python and ML model to use.

*activity_home_page.xml

5.5.1.2 File Handler

This class is utilised by many other classes and handles saving files to the phone given an InputStream. It also implements methods to delete both the audio files and waveform files from the phone, to be used when a recording is deleted to ensure all information about that recording is also removed.

This class is created in the Home Page when the app starts and then is passed to any classes that require it, to ensure that only one copy of it exists at any given instance.

5.5.1.3 JSON Handler

This class is also utilised by many other classes and is only instantiated once (same with the storageDir variable which points to the file location to store any files).

A JSON file is used to store information about each of the recordings so that it is maintained even when the application is closed, or the phone is switched off. This class handles all interaction with that JSON file:

- Creating JSON file when app is first installed
- Loading JSON Object from the file to be used programmatically
- Saving changes back to the JSON file
    - Deleting one or all recordings
    - Getting most recently added recordings
    - Loading more recordings (for infinity scrolling)
    - Adding a recording
    - Adding one or many tags to a recording
    - Deleting one or many tags from a recording
    - Searching/Filtering and returning a list of recordings that match the query
    - Getting the total number of recordings
- Defines constant values for ON/AT/BEFORE/AFTER for filtering so they are the same everywhere

The JSON file has the structure:

```
{ “size”: <number_of_recordings>,
  “recordings”: [<array_of_recording_objects>] }
```

A recording object has the structure:

```
{ “id”: “YYYY-MM-DD-HH-MM”,
  “date”: [YYYY, MM, DD, HH, MM],
  “recording”: “<filename>.wav”,
  “tags”: [<array_of_tag_objects>] }
```

A tag object has the structure:

```
{ “tag”: “<tag_name>”,
  “app”: <boolean> }
```

*5.5.1.4 Home Fragment**

The Home Fragment class (and the rest of the fragment classes) extend the Android Fragment class. These classes are displayed as part of the Home Page between the toolbar and the bottom navigation bar and navigating via the bar switches which Fragment is shown.

The Home Fragment is very simple to aid in usability. It provides a link to each of the other two pages as well as a button to show the device instructions. This currently just displays a ‘Coming soon’ message.

*fragment_home.xml

Home Fragment also implements the Device Observer interface. This class, along with the Device Subject class used the Observer design pattern, explained further in section 5.5.2. In this case it is used to update the device status text on the home page which will indicate if the device is connected or not.

5.5.1.5 Device Observer

Device Observer is an Observer interface from the Observer design pattern. In this case it allows classes to listen for changes to the device, namely when it connects, disconnects, or gets new data.

It should be noted that Connect and Disconnect updates only change an integer value in each of the classes which is then used to set the display text. This is because the Bluetooth runs in a background thread, and you cannot update the view from a thread it was not created in. The drawback of this is that the status text is only changed when the user clicks off and back onto the view.

5.5.1.6 Device Subject

Device Subject is a Subject from the Observer design pattern. It defines the integer value for each of the notify types, and implements the required methods: attach, detach, and notify.

The three types of notify are CONNECTED, DISCONNECTED, and NEW_DATA. For the first two the second parameter (info) is null and for the third type it is a string which contains the filename of the new recording. This implementation uses the ‘push’ update method.

*5.5.1.7 Device Settings Fragment**

This is another one of the Fragment classes and hence extends the Android Fragment class. It handles displaying information about the physical device and taking in user input to manage the device (like connecting, disconnecting, or changing modes). It holds a list of valid modes that the design can be in: Standby, Low Sampling Rate, or Continuous Sampling.

Device Settings Fragment also holds an instance of the Device Controller object which is responsible for all interactions with the physical device, therefore it passes the user interactions on to Device Controller.

Device Settings Fragment implements Device Observer to update the view about the connection status. However, it also handles when a new file is received using the New Data update type. Using the given filename, it uses the JSON Handler to update the JSON file with the new information and then calls the Model Handler to process the file.

When calling the Model Handler, it creates an instance of the Model Callback. The model runs in a background thread, so a callback is needed to update the main thread when it is finished. In this case is passes back the tags that were found by running the model so that they can be added to the relevant recording.

* fragment_device_settings.xml

5.5.1.8 Device Controller

As previously mentioned, Device Controller handles all interaction with the physical device (i.e. BLE). It is a boundary between the physical device and the rest of the app allowing interactions like getting the battery or connection status, as well as calling methods like connect or sync.

Device Controller implements Device Subject as it is the Object that the rest of the app wants to know about. Because of the design of the Observer pattern, it can update the Observers without needing to know anything about them.

It should be noted that many fields/methods related to BLE have been left out of the diagram for brevity as they are not relevant to the rest of the application or its relationships.

5.5.1.9 Model Handler

This class handles everything to do with running the Machine Learning model. This must be done in a background thread as it is very intensive (often taking 10 or more seconds) so would cause unacceptable delays in the view if run on the main thread.

When a file is processed, initially a spectrogram is created from the audio file. This needs to be done in python as the required libraries do not have Java equivalents. The generation of waveform images to later display to the user is also handled by this code. Then the image is pre-processed into the format that the model is expecting. Finally, the tflite model is run on the input and the output is processed. The process of running python from the Android app and running the model is discussed further in section **Error! Reference source not found.**.

The very last action of the Model Handler (when processing a recording) is to call the onComplete method of the callback it was passed.

5.5.1.10 Model Callback

Model Callback is an interface which requires the onComplete method. As mentioned previously, this is used to update the main thread with information when a background thread task is completed.

5.5.1.11 Recordings Fragment*

This is the last of the Fragment classes that extends the Android Fragment class. Recordings Fragment handles displaying everything on the recordings page, it also holds some of the 'sub-views' that are only displayed some of the time (like the filter tab or the waveform display).

The Recordings Fragment also uses the toolbar at the top of the screen to condense some of the additional functionality:

- Search: while typing searches the currently loaded items, when enter is pressed it uses the JSON Handler to search all recordings. It displays a loading symbol while typing to indicate more data can be loaded.
- Delete All: shows a confirmation message then if confirmed, deletes all recordings. This uses the JSON Handler to remove all information stored in the JSON and the File Handler to remove all audio and waveform image files.
- Refresh: uses the JSON Handler to get the most recent items then displays them. If new recordings are available this will show them. If the model has processed a recording this will update the tags.
- Download all: since all recordings are available on the phone this currently does nothing. In the future, this could be changed to extract all files to a zip and put them in the downloads folder (for example).

- Filter: shows the filter sub-view which calls the JSON Handler filter function with the parameters set by the radio buttons and date/time pickers.
- Sort: calls the Recording List View Adapter sort method (discussed further in section 5.5.1.15).

Recordings Fragment uses an 'onScrollListener' to implement 'infinite scrolling', i.e. load more data as the user scrolls. This must be disabled when searching/filtering otherwise it tries to add more entries to the end of the results which incurs unpredictable behaviour.

*fragment_recordings.xml

5.5.1.12 Recording

Recording is an Object representation of a recording that is used when recording info is loaded from the JSON, or when recordings are used within the application in general.

It stores filename, date and time, ID, and an ArrayList of tags.

5.5.1.13 Tag

Similar to Recording, Tag is an Object representation of a tag. It stores the String value of the tag along with a Boolean of whether the tag was added by the user or the application.

5.5.1.14 Date/Time Picker Fragments

These are used to display a calendar or clock for the user to enter dates and times. They both extend the Android DialogFragment class and implement the Date/Time PickerDialog. This takes existing framework for displaying date/time selectors and adds minor functionality to aid in use with the rest of the app. For example, setting the filter view TextView when a date/time is selected and storing the date/time selected to be retrieved later.

5.5.1.15 Recording List View Adapter

Recording List View Adapter extends the Android class: RecyclerView.Adapter<RecyclerView.ViewHolder).

A RecyclerView is an Android view class which displays a list of view items to the screen. The adapter holds a data list, has a view class which maps this data to a sub-view, and handles any changes to the data list. In this case, Recording List View Adapter holds a list of Recording objects, interfaces with the rest of the app to perform operations on them, then updates the RecyclerView accordingly. When the RecyclerView displays an Object from the data list:

- if the current item is a Recording, it will display the corresponding view with its data
- if it is null, it will display a loading symbol

This behaviour had to be specifically encoded.

Any time a recording is added/deleted/changed the List View Adapter calls a notify method which lets the RecyclerView know to change what is displayed. It does this automatically based on what is in the data list. Because the view is based directly on what is in the data list, a copy is used for when temporary operations (like filter/search) are performed. In these cases, the data list is altered so that the view can update but the copy keeps the full list so the data list can be restored when the temporary operation is done.

*5.5.1.16 My View Holder**

My View Holder extends the Android class RecyclerView.ViewHolder, to be compatible with the List View Adapter. This class directly corresponds to a single recording view in the display and is what the RecyclerView uses to map a Recording Object from the data list to a view to display. It also handles any interactions with the single recording sub-view. This includes:

- Playing the recording audio (discussed further in section 5.5.1.18)
- Displaying the waveform image of the recording
- Displaying the options view:
    - Download: displays a message stating where on the phone the recording can be located
    - Manage Tags: displays the manage tags view**
    - Delete: deletes the individual recording using the JSON Handler and File Handler

**The Manage Tags view uses its own RecyclerView and hence another ListViewAdapter, discussed further in section 5.5.1.21. The add tags function lets users pick from a list, not enter their own. The delete tags function lets users delete any tags they added, not tags added by the application.

*recording_view_item.xml

*5.5.1.17 Loading Holder**

Loading Holder also extends the Android class RecyclerView.ViewHolder. This class is responsible for displaying a loading icon when a item in the data list is null. This is used to display a loading icon if the infinite scrolling takes too long to load and with the search function.

*recording_item_loading.xml

5.5.1.18 List Media Player

List Media Player extends the MediaPlayer Android class. It acts as a wrapper to use the default media player while handling the fact that different audios from the list of recordings are trying to use it. It ensures that two recordings cannot play at once by storing the id of the currently playing audio.

The default MediaPlayer runs in a background thread by function so as not to inhibit the view.

The List Media Player used the Observer pattern again to set the play/pause icon on the button when the recording ends or when it is paused by playing another recording. This is necessary as these events are triggered out with the recording itself.

5.5.1.19 Play Pause Subject

Play Pause Subject is a Subject from the Observer design pattern. In this case it is implemented by the List Media Player which notifies all observers whenever a play/pause occurs. This takes the form of the String id of the audio that it currently has along with a Boolean paused. When an Observer receives the notification, it will only update if the id matches and then will change its view based on the paused Boolean.

5.5.1.20 Play Pause Observer

Play Pause Observer is an Observer from the Observer design pattern. In this case it is implemented by all instances of My View Holder as their image buttons need to change between a play and pause icon when a play/pause occurs. When the button is clicked this is easy and doesn't require the pattern, however the Observer pattern lets the icon update when:

- A different recording is played (forcing this to pause)
- The recording ends

These cases are both impossible without an Observer or other solution.

*5.5.1.21 Tag List View Adapter**

Similar to Recording List View Adapter but for displaying the list of tags. It stores a list of valid tags as well as a list of unused tags (valid tags minus used ones), to display the list of tags that can be added.

It also stores a Recording Object because each individual Recording has its own Tag List View Adapter.

It has two view types (similar to the Recording version as well). One is used for displaying application generated tags and one for displaying used added tags. This ensures that users cannot delete tags that were generated by the application via running the model.

*uses manage_tags_view.xml, tag_app_added.xml, tag_user_added.xml

5.5.2 Observer/Listener Design Pattern

A Software Design Pattern is a generalised solution to a common problem within the field. In this case the Observer (AKA Listener) Design pattern is utilised. Its purpose is to define a one-to-many relationship between objects where when one object changes state its dependants are notified and updated automatically. It maintains consistency between related objects while keeping them loosely coupled. The relationship diagram for the pattern is shown in Figure 19.

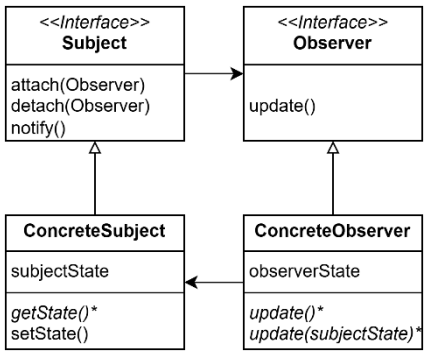


Figure 19. Observer Design Pattern diagram

The interfaces define the methods that any concrete classes must implement. The notify method in the Subject will go through each observer is has and call its update() method. The notify method should always be the last thing a Subject does to ensure that it is in a steady state.

The pattern has two models: Push or Pull. In a Pull model, the update method has no parameters, and the Observer will call subject.getState() within. In a Push model, the Subject pushes information to the observers by passing its state as parameter(s) of the update() method. Both instances in this application used the Push model.

The subjectState and setState() methods should be considered placeholders for the Subject's variables and methods that will change them.

5.6 INTERACTION WALKTHROUGHS

Below are a few examples of user use cases and the interaction path through the application. For each demonstration the following recordings were used with sample time stamps:

Table 10. List of recordings and tags for demonstration

Recording Name/Time Stamp	Tags
2024-11-09-15-47-59	Normal
2024-12-03-06-20-34	Abnormal
2025-01-01-14-55-06	Abnormal, Walking
2025-01-01-14-56-43	Abnormal
2025-04-12-17-01-12	Abnormal
2024-10-11-22-19-04	Abnormal, Resting
2024-11-29-16-20-46	Unknown
2025-03-23-09-22-01	Unknown

2025-03-23-09-22-48	Unknown
2025-03-23-09-23-13	Normal

5.6.1 Connect, Record, Disconnect

To connect to the device, record some samples, then disconnect from the device the user would follow the steps given below. The first few steps are illustrated in Figure 20, the full walkthrough can be found at Appendix C.

Steps:
1: Click to device settings
2: Click connect (if not automatically connected)
3: Click mode dropdown
4: Select recording mode
wait until ready to stop recording
5: Click mode dropdown
6: Select Standby mode
7: Click disconnect
8: Click to recordings
9 & 10: Refresh to see new recordings (if necessary)

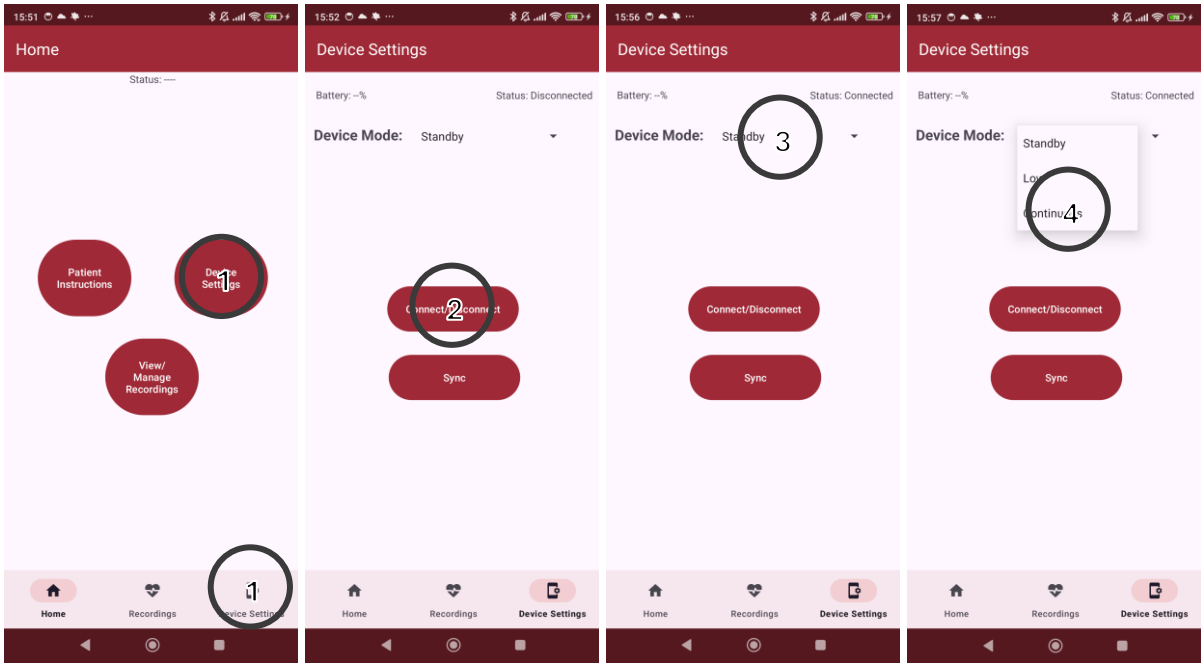


Figure 20. User Walkthrough for connect-record-disconnect

5.6.2 Filter by Date/Time, Add Tag

An example interaction of a user filtering to find a recording from January 1st 2025, then adding a 'walking' tag would follow the steps given below. The first few steps are illustrated in Figure 21, the full walkthrough can be found at Appendix C.

Steps:
1: Click Filter icon
2: Select 'On' date filtering mode
3: Click to select date
4 & 5: Select date from calendar
6: Click Filter
7: Close filter menu (by clicking icon or clicking outside window)
8: Click Options on the relevant recording

9: Click Manage Tags
10: Click Add New Tag
11: Select ‘walking’ tag

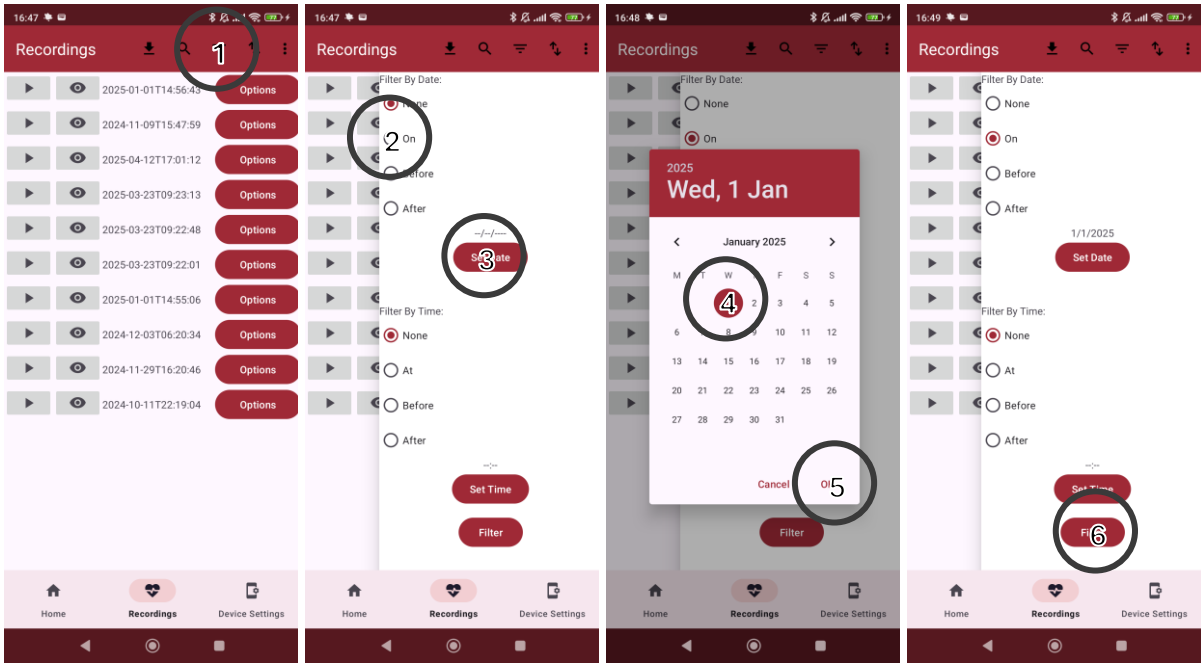


Figure 21. User Walkthrough for filtering and adding a tag

5.6.3 Search by Tag, View Waveform

An example interaction of a user searching for all recordings with the tag ‘walk’, and then viewing one of the waveforms would follow the steps given below. The first few steps are illustrated in Figure 22, the full walkthrough can be found at Appendix C.

Steps:
1: Click Search icon
2: Type search query (while typing searches loaded recordings)
3: Click enter (searches all recordings stored on phone)
4: Click view icon on relevant recording
5: Click anywhere outside waveform view to close

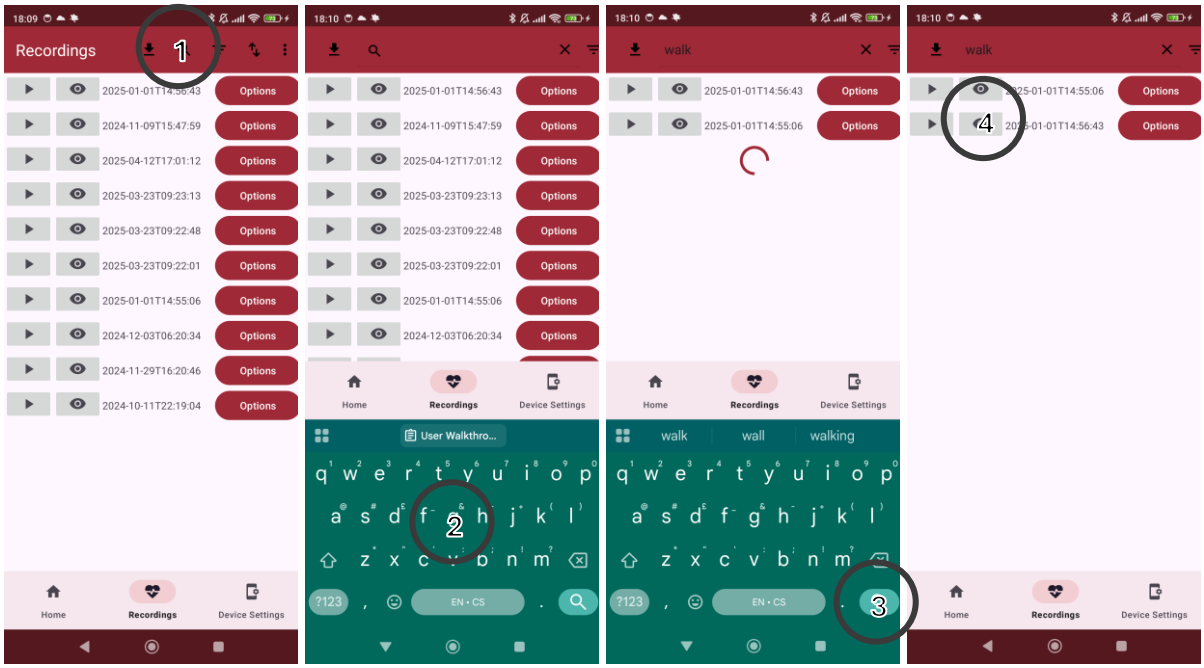


Figure 22. User Walkthrough for searching and viewing a waveform

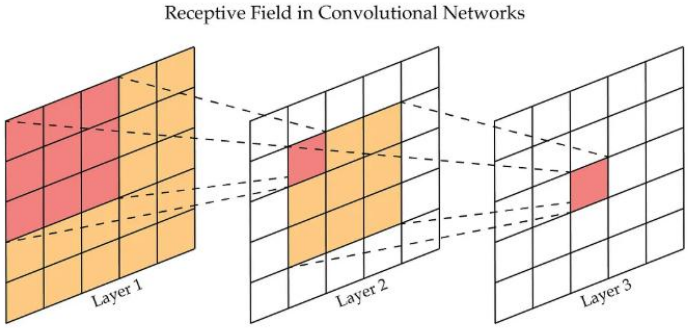
6 MACHINE LEARNING MODEL

6.1 DATA HANDLING

6.1.1 CNNs for Image Classification

Perception problems - such as detecting items in images, or words in speech - are generally easy tasks for humans, but very difficult tasks for computers. Humans take advantage of sensory input to identify and detect patterns; Convolutional Neural Networks are an attempt to harness this by replicating the structure of the brain's visual cortex. This structure is built on convolution operations and allows CNN models to efficiently recognise patterns and key features in images, making them ideal for image classification, object detection, and video segmentation tasks.

Like in the visual cortex, neurons in the CNN each have a small local receptive field, which refers to the specific region of the input image that neuron is connected to. Each neuron is programmed to respond to specific patterns in its local field depending on the classification problem. As local receptive fields overlap, they build up into the full visual field of the input image. In this way, as signals move through layers of local fields, the connected neurons respond to steadily more complex patterns.



The receptive field essentially determines the amount of information a neuron has when it makes a prediction. Size of the field can be adapted through parameters such as stride and kernel size to suit the specific application. Therefore, the main factors to consider when making a CNN are the type of image being delivered as input, the hyperparameters determining receptive fields of each layer, and the actual number of convolutional, pooling and fully connected layers that make up the model.

6.1.2 File Pre-Processing

6.1.2.1 Filtering and Denoising

While most of the dataset files were reasonably clean and clear signals, some contained significant noise. Filtering was first applied to clean up the dataset audio files before they were plotted as spectrogram images for model input. Different iterations of filtering were tested and the final iteration put the audio files through a fourth-order 25-400 Hz Butterworth bandpass filter implemented with scipy's .signal library. Normal heart sounds typically fall between 20-200 Hz [47], while abnormal can fall anywhere below 500 Hz but are most common between 25-400 Hz.

```
def bandpass_filter(input_wav, sr):
  wlow = 25/(sr/2)
  whigh = 400/(sr/2)
  b, a = signal.butter(4, [wlow, whigh], btype='band')
  output_wav = signal.filtfilt(b, a, input_wav)
  return output_wav
```

The signal was then passed through a pre-emphasis filter, essentially a high-pass filter intended to boost frequencies at the higher end of the 25-400 Hz bandpass.

```
def pre-emphasis(x, alpha=0.97):
  return np.append(x[0], x[1:]-alpha*[x:-1])
```

Below is a plot of an example audio file taken from the training dataset, plotted to a waveform at each stage of the filtering.

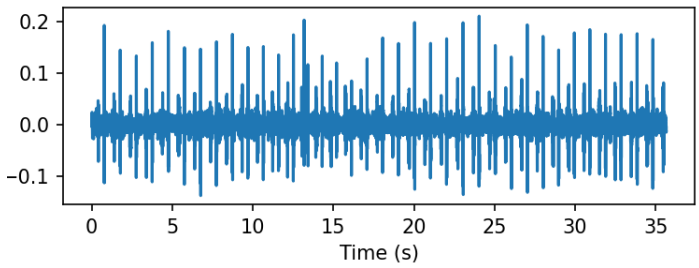


Figure 23. Dataset recording a0001 - Original Waveform

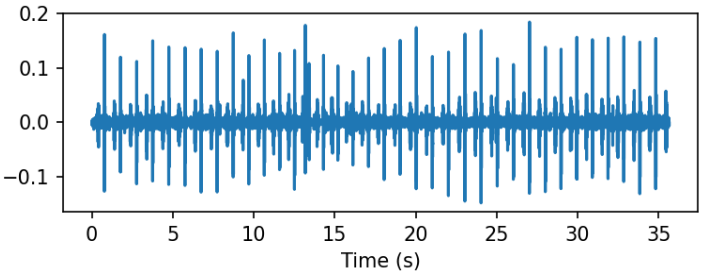


Figure 24. Dataset recording a0001 – Band-passed Waveform

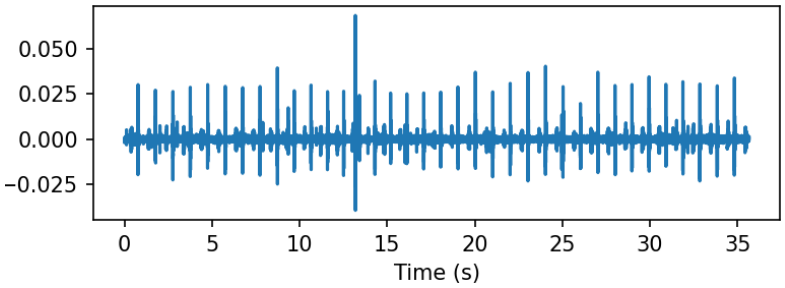


Figure 25. Dataset recording a0001 - Band-passed and Emphasised Waveform

The above plots are of recording a0001 from the test set, classified by the CHSR database as abnormal. By the time both filters have been applied in Figure 25, the abnormality in the heart sound can be identified visually as the peak that occurs at around 14 seconds.

6.1.2.2 File Durations

One of the issues seen in literature [48] was that the duration of dataset files could vary anywhere between 2 and 120 seconds. For ideal ML performance, all the data should be as closely related as possible, with the only difference between the files being the abnormalities if/when present. If spectrograms were generated regardless of duration, files toward the shorter end would appear to have 'higher' resolution, while files on the longer end would appear compressed or squashed. This would make it difficult for the model to correctly identify abnormalities.

In literature, this problem was avoided by segmenting all dataset files in 2 or 5 second chunks. However, none of the datasets (either in literature or in this application) included labelling of exactly when an abnormal heart sound appeared in a recording. Splitting all files into 2 second chunks therefore ran the risk of segments without an abnormality (due to splitting) being labelled as abnormal. For example, a 10 second file with one heart abnormality 2 seconds in would result in five files labelled 'abnormal', but only one of those files would contain an abnormal heart sound. The literature referenced above gave no guidance on how they addressed this issue.

The solution decided on was to select a set duration that most dataset files were reasonably close to and either pad the file through repetition if it was shorter, or trim it if longer. Care had to be taken with abnormal files – as trimming could remove a singular abnormality.

- Normal file < Duration: Pad by repeating
- Normal file > Duration: Trim
- Abnormal file < Duration: Pad by repeating
- Abnormal file > Duration: Discard file to preserve classifications

When analysing the durations of all files in the dataset, the average normal file was 21.9s and the average abnormal 24.8s. The mean between these two numbers (23.35) was doubled, then rounded down to 45s. A set duration of 45s meant that on average, each short file only needed to be repeated once for both normal and abnormal, and only 3.3% of the abnormal dataset was discarded. An example of an extended abnormal file is shown below.

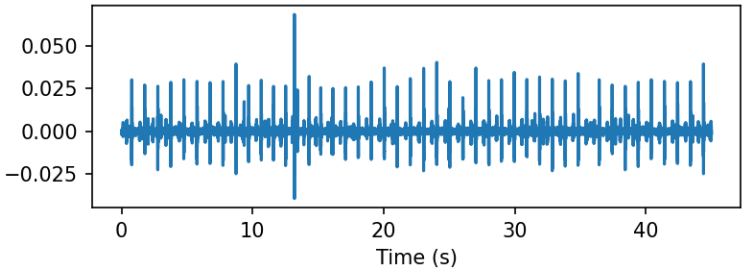


Figure 26. Dataset recording a0001 - Band-passed and Emphasised Waveform extended to 45s

6.1.2.3 Spectrogram Image Transformation

From input type validation, the input images took the form of Mel-frequency cepstrum coefficients (MFCC) spectrograms. MFCCs are widely used to extract key features from audio data for speech processing. They are a set of coefficients that capture the shape of the power spectrum of a sound signal, discarding less useful information and only preserving the most relevant frequency components of a signal. Python's librosa library includes tools for generating MFCCs that automatically carry out all this computation. A useful tool can be first plotting the mel-frequency spectrum, then extracting the MFCCs from that.

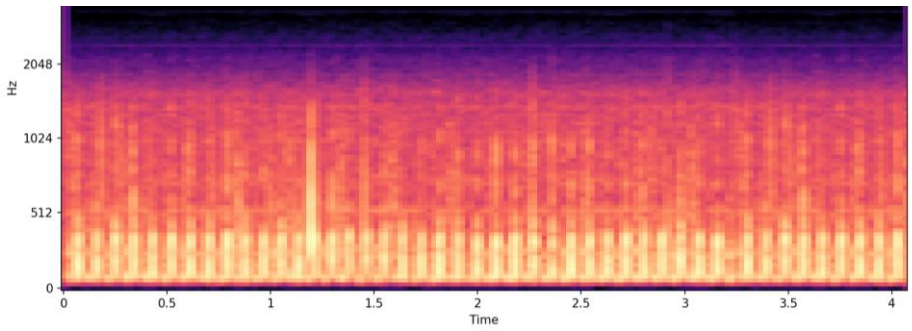


Figure 27. Dataset recording a0001 - Filtered mel spectrogram

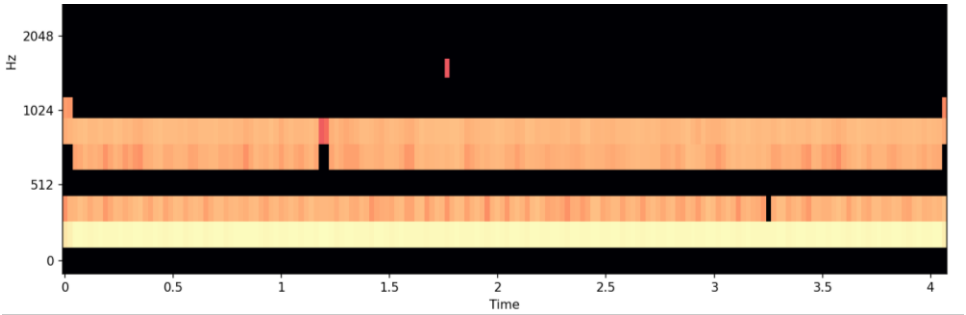


Figure 28. Dataset recording a0001 - Filtered MFCC spectrogram

This process was repeated for all audio files in the dataset, with images in the MFCC format (Figure 28) serving as model input.

6.1.3 Data Augmentation

6.1.3.1 Dataset Split

The Classification of Heart Sounds Database (CHSR) from the 2016 PASCAL Challenge contained 3126 audio recordings total. The chosen dataset split for training, validation and test sets was 80-10-10, a standard split for image classification challenges with a small number of classes.

- Training set = 80% = 2500 recordings
- Validation set = 10% = 313 recordings
- Test set = 10% = 313 recordings

Some model iterations experimented with different dataset splits to compensate for the relatively small dataset size – such as 60-20-20 – but it was found that increasing the overall dataset size through augmentation techniques was more effective than just adjusting the split.

6.1.3.2 Class Balancing

Of the total 3126 audio recordings, 2481 (75%) were classified as normal and 645 (25%) as abnormal. As the classes were unbalanced, there was a risk of them being distributed unevenly in the training, validation and test splits and therefore skewing the model results. To fix this, classes were first stratified when split into the three datasets so that each dataset had the same ratio of 75% normal files to 25% abnormal files. This could still make it difficult to get the true performance of a model (e.g. with 75% normal files, a model that predicts normal every time will still report an accuracy of 75%), so abnormal files were up sampled in each dataset until they matched the number of normal files in an even 50-50 split.

- Training set = 2500 total = 1984 normal: 516 abnormal = 1984 normal: 1984 abnormal
- Validation set = 313 total = 248 normal: 65 abnormal = 248 normal: 248 abnormal
- Test set = 313 total = 248 normal: 65 abnormal = 248 normal: 248 abnormal

To create the missing abnormal files needed to achieve that 50-50 split, general data augmentation techniques such as rotating, resizing or editing the colour scheme of an image could not be used. Instead, normal files were chosen and spliced at random, then combined with abnormal files. This was repeated until class balance was reached.

6.2 MODEL ARCHITECTURE

6.2.1 Base Architecture

To give the model the best possible chance to integrate well with the PCB and physical device, the first step of the model design was to find a suitable, pre-existing CNN architecture on which to base any future structure edits and hyperparameter tuning. To identify possible base CNN architectures, it was useful to look at the winners of the annual Large Scale Visual Recognition Challenge (ILSVRC). The challenge is image classification on 256x256 images belonging to 1000 different classes and is a popular testing ground for new CNN models.

AlexNet, GoogLeNet, VGGNet, ResNet, Xception and MobileNet architectures were all considered. The main issue with the above models was the resources needed to train and compile them, with training times often exceeding four hours and memory requirements exceeding what the computer could allocate (Windows overcommitment problem). The earlier models of AlexNet and GoogLeNet were particularly resource-expensive. MobileNet was promising because it is a model designed explicitly to be fast and lightweight for deployment in mobile and embedded system applications, but during testing it struggled with the higher resolution needed by mel-frequency spectrograms.

The selected base architecture was ResNet (Residual Network), a set of deep models with a variable number of layers (e.g. ResNet-34, ResNet-50, ResNet-152). The key to the depth of ResNet models is the use of skip connections, in which the signal at the input of a layer is also added to the output of a layer higher up the stack.

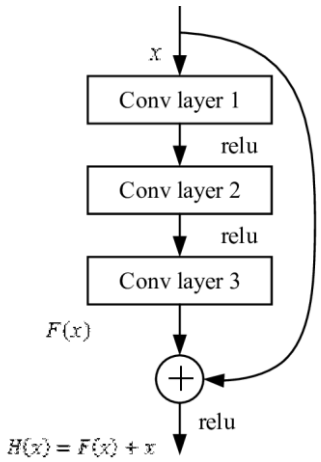


Figure 29. Skip connection diagram

This dramatically speeds up training because it allows the model to make progress even if several layers have not started learning yet. Smaller networks such as ResNet-18 and ResNet-34 are well-suited for real-time applications with limited resources, such as on a mobile device. In literature, ResNets work well across image classification problems in medical imaging, with layer depth as a trade-off between speed and accuracy.

6.2.2 Model Structure and Summary

Before hyperparameter tuning in integration testing, the CNN followed a ResNet-34 architecture. 34 layers was enough depth for the model to comfortably train and compile in under an hour on the university computers, while also keeping accuracy above 85% on the unseen test sets.

A wrapper class was used to define the standard convolutional layer.

```
DefaultConv2D = partial(tf.keras.layers.Conv2D,
  kernel_size = 3,
```

```
  strides = 1,
  padding = “same”,
  kernel_initializer = “he_normal”,
  use_bias = False)
```

The kernel size was set as 3x3 because the input images are not large (256x256) to reduce storage on the application. Padding as ‘same’ keeps the spatial dimensions the same through the layers.

A class was then used to define a single residual block – this allowed the block to be reused by the model and instantiated as many times as needed. Having the residual block as its own class also made it easier to loop through different configurations later in the model structure.

```
class ResidualUnit(tf.keras.layers.Layer):
  def __init__(self, filters, strides=1, activation =“relu”, **kwargs):
     super().__init__(**kwargs)
     self.activation = tf.keras.activation.get(activation)
     self.main_layers = [
            DefaultConv2D(filters, strides=strides),
            tf.keras.layers.BatchNormalization(),
            self.activation,
            DefaultConv2D(filters),
            tf.keras.layers.BatchNormalisation()
     ]
```

The main path (main_layers) applies the convolution. As an initial stem, the standard ResNet stem of a 7x7 convolutional layer was used. The rest of the model followed ResNet-34s layer configuration:

- 3 residual units with 64 filters
- 4 residual units with 128 filters
- 6 residual units with 256 filters
- 3 residual units with 512 filters

Each residual unit automatically built skip connections if needed. The full model summary is shown below in Figure 30.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 64)	9,408
batch_normalization (BatchNormalization)	(None, 128, 128, 64)	256
activation (Activation)	(None, 128, 128, 64)	0
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0
residual_unit (ResidualUnit)	(None, 64, 64, 64)	74,240
residual_unit_1 (ResidualUnit)	(None, 64, 64, 64)	74,240
residual_unit_2 (ResidualUnit)	(None, 64, 64, 64)	74,240
residual_unit_3 (ResidualUnit)	(None, 32, 32, 128)	230,912
residual_unit_4 (ResidualUnit)	(None, 32, 32, 128)	295,936
residual_unit_5 (ResidualUnit)	(None, 32, 32, 128)	295,936
residual_unit_6 (ResidualUnit)	(None, 32, 32, 128)	295,936
residual_unit_7 (ResidualUnit)	(None, 16, 16, 256)	920,576
residual_unit_8 (ResidualUnit)	(None, 16, 16, 256)	1,181,696
residual_unit_9 (ResidualUnit)	(None, 16, 16, 256)	1,181,696
residual_unit_10 (ResidualUnit)	(None, 16, 16, 256)	1,181,696
residual_unit_11 (ResidualUnit)	(None, 16, 16, 256)	1,181,696
residual_unit_12 (ResidualUnit)	(None, 16, 16, 256)	1,181,696
residual_unit_13 (ResidualUnit)	(None, 8, 8, 512)	3,676,160
residual_unit_14 (ResidualUnit)	(None, 8, 8, 512)	4,722,688
residual_unit_15 (ResidualUnit)	(None, 8, 8, 512)	4,722,688
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 2)	1,026

Figure 30. ML Model Summary (ResNet-34)

The final Dense layer is responsible for converting the spatial output used through the model into a flat feature vector. At this stage, it classifies the output into either ‘normal’ or ‘abnormal’, and the results are passed on to the application.

6.3 Training Results

The CNN was given a standard learning rate of 0.0001. It was trained on the full training set from the CHSR database, with 20% kept aside for validation and test sets. Datasets were separated in the original directories to ensure no leakage when imported in the code. The best performing model using dataset files only (i.e. no hardware integration yet) was trained on files up-sampled to 2 kHz, put through a fourth-order Butterworth 25-400 Hz bandpass filter, then converted into MFCC spectrograms. The model was evaluated on the unseen test set of identically generated images. Accuracy and loss for training and validation sets were plotted against each other below in Figure 31.

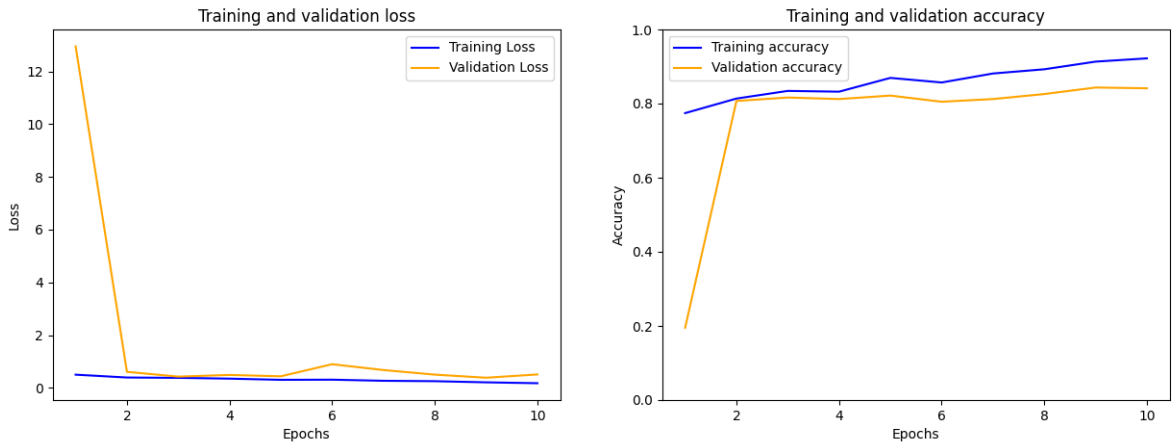


Figure 31. ML Accuracy and Loss Graphs

These graphs corresponded to an accuracy of 93.62% on the training set, 84.63% on the validation set, and 93.90% on the test set, as seen in Figure 32.

```
COMPILED
Epoch 1/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 519s 6s/step - accuracy: 0.7549 - loss: 0.5536 - val_accuracy: 0.8121 - val_loss: 0.6497
Epoch 2/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 459s 6s/step - accuracy: 0.8137 - loss: 0.3992 - val_accuracy: 0.8121 - val_loss: 1.0525
Epoch 3/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 419s 5s/step - accuracy: 0.8221 - loss: 0.3768 - val_accuracy: 0.8152 - val_loss: 1.0647
Epoch 4/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 417s 5s/step - accuracy: 0.8569 - loss: 0.3166 - val_accuracy: 0.8261 - val_loss: 0.4117
Epoch 5/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 432s 5s/step - accuracy: 0.8623 - loss: 0.3018 - val_accuracy: 0.8292 - val_loss: 0.6725
Epoch 6/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 456s 6s/step - accuracy: 0.8735 - loss: 0.2624 - val_accuracy: 0.8478 - val_loss: 0.3629
Epoch 7/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 522s 6s/step - accuracy: 0.9111 - loss: 0.2040 - val_accuracy: 0.8416 - val_loss: 0.3266
Epoch 8/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 524s 6s/step - accuracy: 0.9192 - loss: 0.1991 - val_accuracy: 0.8432 - val_loss: 0.3717
Epoch 9/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 423s 5s/step - accuracy: 0.9262 - loss: 0.1715 - val_accuracy: 0.8354 - val_loss: 0.5475
Epoch 10/10
81/81 ━━━━━━━━━━━━━━━━━━━━ 519s 6s/step - accuracy: 0.9362 - loss: 0.1645 - val_accuracy: 0.8463 - val_loss: 0.4414
81/81 - 109s - 1s/step - accuracy: 0.9390 - loss: 0.1329
0.9390290975570679
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
[[0.19 0.81]
 [0.14 0.86]
 [0.99 0.01]]
[1 1 0]
['Abnormal' 'Abnormal' 'Normal']
```

Figure 32. Terminal output for ML Model

7 PHYSICAL HOUSINGS

7.1 BRACE

The physical mechanism for attaching the sensing device to the body is crucial in determining its suitability for long-term continuous use in terms of ease of use, comfort and durability as well as providing a stable & secure attachment to reduce noise and motion artifacts. Wearable designs for other heart monitoring devices consist of chest straps, vests and adhesive patches.

Chest straps were found to have a much higher accuracy in monitoring heart rate than vests during exercise, with vests performing adequately only for a certain athletic body type [49]. This discrepancy might be due to unstable contact between electrodes and skin arising from ill-fitting vests. However, another ECG-based heart rate monitor that used a chest strap was found to be unsuitable for individuals with higher upper body obesity due to it slipping down [50]. Stable and firm contact of sensors with the skin is a key component of ensuring reliable data. In designing a securing device for the sensor, it was integral to offer a level of adjustability and custom fit to ensure contact for a wide range of body types and sizes, especially considering that those at risk of heart problems tend to have body types out with the standard athletic body type catered to by fitness devices.

Some chest electrode sensors or other accelerometer-based or microphone-based sensors are available as an adhesive patch [12]. This will require additional adhesive gels which can cause skin irritation, fall off and cause damage when adhesive quality falls or meets an unexpected force and crucially for our device, cannot provide sufficient pressure for reliable readings. While an adhesive could be used with the device, creating a brace was essential to prove ease of use and to provide a secure fit suitable for long-term use.

The main design criteria identified were comfort, adjustability, suitability for a range of body types, and ensuring firm contact with the skin. Several designs were proposed as detailed in Table 11 and shown in Figure 33.

Figure 33. Brace iterations 1, 2, and 3

Table 11. Brace design iterations

	Description	Comments
Design 1	A rectangular patch to cover the device with a thick elastic underneath to hold brace against body and a thinner elastic wrapping around the device itself to ensure sensor contact	Device doesn't maintain contact with skin during spinal flexion Brace slips down when hands raised Elastic bands would tangle & twist & were visually unappealing
Design 2	Triangular fabric that covers the device and connects to a shoulder strap to prevent brace from sliding down + apply pressure to the device Thick adjustable elastic underneath to hold brace against body	Does not work on female bodies as it doesn't provide enough pressure to the device
Design 3	Fabric to wrap around half the body with sewn-in channels to maintain distance between elastic straps Adjustable shoulder strap to prevent device from slipping down, applying tension from above Thin elastic wrapping around the device itself to ensure sensor contact Thick adjustable elastic underneath to hold brace against body	Works for both male and female body types Elastic straps twist inside the channels creating hassle
Design 3.1	Foam added within channels to provide structure	Elastic straps no longer twist inside channels Brace has a better structure & comfort

The final design consists of the following, numbered on Figure 34:

1. A thick elastic strap that is secured horizontally around the waist, just underneath the device. This is the main support for fastening the brace to the body and holding the device
2. A secondary elastic strap is fastened horizontally across the body and the device, ensuring firm pressure of the device against the skin.
3. The main fabric body, which consists of two channels for the elastic straps. The channels are sewn with a distance corresponding to the distance between the bottom of the device and the MEMS position on the device to eliminate user error in positioning the elastic straps over the device. This ensures that the secondary strap is always over the sensor. The channel has foam built-in to provide structure and slight rigidity whilst still being comfortable.
4. An adjustable shoulder strap which connects to the main fabric body. One end of the strap has a triangular shape designed to cover the device, while the other end of the strap attaches at the back. This prevents the brace from slipping down or rotating around the torso, as well as providing tension from above to keep the device in place.

Figure 34. Final brace design on human

Nylon was selected to due to its moisture-wicking properties, stretch, and durability. A braided elastic band was chosen for the main horizontal band as it has flexibility for comfort but doesn't stretch out over time, and has a large width to prevent it from digging in. It has a buckle that user adjusts around chest until it is snug. There is another piece of smaller elastic that fits around the enclosure where the microphone is in contact with the skin, applying pressure at all times. The thick elastic band and nylon cover suspends device in place while the upper elastic ensures that it stays in place & prevents minor movements.

The microphone must be at mitral point i.e. just below curvature of Pectoralis Majoris on the left. Thus, PCB designed so that the microphone is at the top, and the device is flat against the ribs.

7.2 ENCLOSURE

The enclosure was designed (in Tinkercad) specifically to fit the PCB and a battery. Like the brace and PCB, the enclosure underwent several iterations while being integrated with the rest of the system.

Due to the nature of the device being placed directly between the brace and the user's skin considering the shape of the enclosure was vital. To slide easily inside the brace and allow the user to move freely without compromising the device's contact the considered shape of the enclosure was a flattened half dome with the flat baseplate sitting against the user.

7.2.1 Enclosure – 1st iteration

If a fully circular dome was used a PCB size of 52.5 mm x 40.5 mm would require the enclosure to have a diameter of 71 mm. To mitigate this, the first iteration of the enclosure was a cuboid rounded on the sides and top corners. The height of the enclosure was kept to a minimum only accounting for the thickness of the PCB and a single watch cell battery. The enclosure features 4 screw holes designed to be screwed from top down to the base plate. The screw shafts were designed to be shorter than the enclosure itself to allow for the thickness of the PCB. The base plate is completely flat except 4 mounting points designed to have threaded inserts heat pressed into them after it is printed. The premise of the design was to place the PCB over the mounting points and place the top shell over top. When screwed down the screw shaft would apply force down onto the PCB keeping it in firm and constant contact with the base plate of the enclosure. This is an important function of the enclosure that allows the MEMS microphone to stay as close to the skin as possible. In this iteration exact measurements, such as screw size, thread size, and mounting point position were not taken into consideration. The purpose of this iteration was purely just to visualize the proof of concept before designing a more

detailed enclosure. This initial design can be seen in Figure 35 and Figure 36 with supporting structures on the underside of the top enclosure (highlighted in red) to improve rigidity of the design as well as supports for the screw shafts (highlighted in green) to make them stiffer and less flexible.

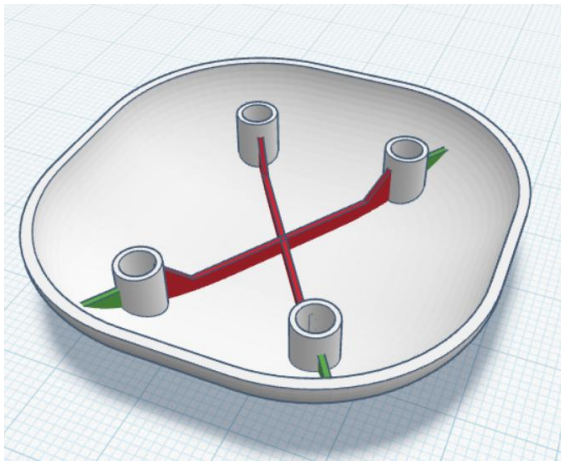


Figure 35. Initial proof of concept design (isometric view)

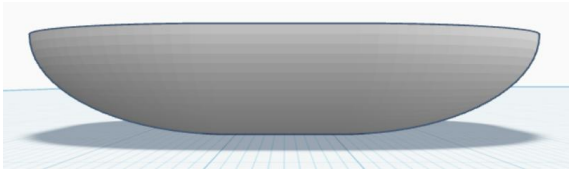


Figure 36. Initial proof of concept design (side view)

7.2.2 Enclosure – 2nd iteration

The second iteration of the design, shown in Figure 37, Figure 38, and Figure 39, focused on more specific measurements of the enclosure as well as modifying the dimensions of the enclosure to fit the updated size of the PCB. The new maximum size of the PCB was set as 47 mm x 25 mm x 20 mm. Therefore, the base of the enclosure was set to a size of 60 mm x 60 mm with rounded corners and a thickness of 1 mm. The PCB was centred onto the base plate and the mounting holes were placed at a relevant distance outside the PCB. The mounting points with the threaded inserts were designed to be compatible with M2 screws. The new height of the shell was 26 mm to account for the height of the PCB and the curvature of the enclosure giving an overall height of 27 mm. The rigidity supports were improved for both the underside and the screw shafts to better fit the new dimensions. The thickness of the top half of the enclosure was set at 1 mm. The screw shafts in the top of the enclosure were designed to fit over its counterparts on the bottom half with a standard clearance of 0.2 mm. A shelf was also placed inside the screw shafts to hold the head of the screw, allowing the top of the enclosure to be tightened onto the bottom. A cutout was also placed in the bottom base plate aligned with the MEMS microphone.

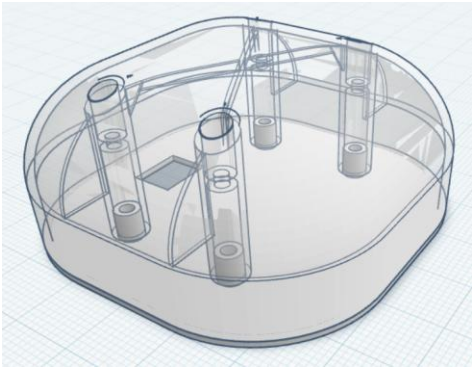


Figure 37. Second iteration enclosure design (isometric view)

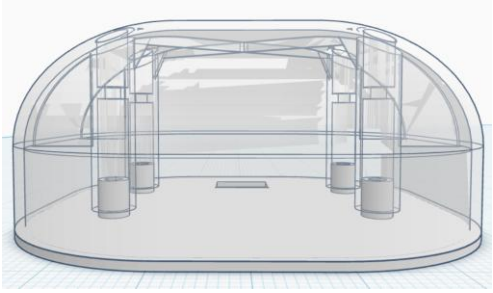


Figure 38. Second iteration enclosure design (side view)

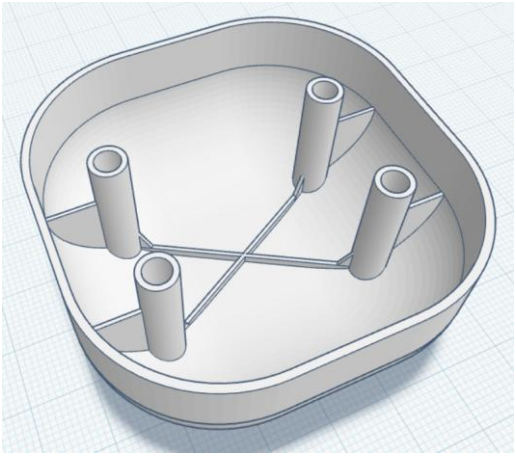


Figure 39. Top shell underside (isometric view)

7.2.3 Enclosure – 3rd iteration

The third iteration accounted for further changes to the PCB with a height of 34 mm instead of 20 mm. The solution was to not change the curvature of the enclosure but to simply make the entire enclosure taller by extending the non-curved part. The screw shafts were also redesigned in this iteration making them sturdier and increasing the tolerances between parts to 0.4 mm as during a test print of the second iteration it was found that 0.2 mm clearance was not sufficient. This iteration also introduced a clasp on top of the enclosure to hold it to the brace. Due to the additional height of the enclosure, structural supports were also placed on the inner walls of the enclosure to reduce warping in all directions.

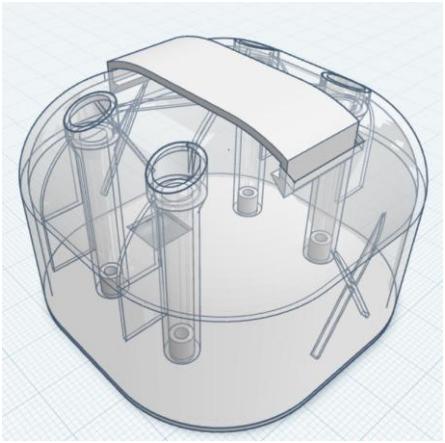


Figure 40. Third iteration enclosure design (isometric view)

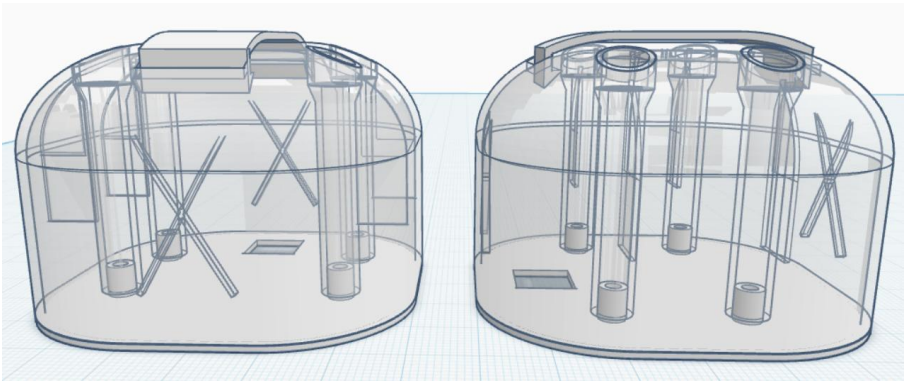


Figure 41. Third iteration enclosure design (side view)

7.2.4 Enclosure – Final iteration

The fourth and final iteration of the enclosure included significant changes its design and size. With a new battery selected (14 mm x 14 mm x 50 mm cylinder) the height of the PCB decreased to a single board with a height of 1.5 mm and a maximum component height of 5 mm. Therefore, the footprint of the device was minimised to 50 mm x 70 mm and an overall height of 26 mm. This iteration kept the original 4 mounting points in the same location, ensuring no changes were necessary to the existing PCB. However, a 5th mounting hole was added at the bottom of the enclosure to ensure stability and adhesion but this was outside of the PCB area.

A battery enclosure was designed to support the battery from the bottom. The design used spacers to sit on top of the base plate mounting points to allow the battery enclosure to sit securely on top without touching any of the PCB components. Ideally the spacers and battery enclosure would be printed as one piece, however due to the limitations of FDM printing (discussed in Section 7.2.5) they had to be printed as separate components. The underside of the shell included extrusions to hold the battery in place from the top. The battery enclosure along with the top supports were designed with no clearances relying on the slight flexibility to provide an ideal fit with no movement or rattling from any parts.

7.2.5 Design Considerations

During the design phase, several considerations had to be made in response to the constraints of Fused Deposition Modelling (FDM) 3D printing. The top piece of the enclosure featured a pronounced fillet, intended to achieve a smooth, rounded finish both for aesthetic purposes and to reduce potential snagging points. However, this geometry significantly reduced the surface area in contact with the build plate, leading to limited adhesion during printing. To mitigate this, a large brim of 15 mm was applied around the first layer to improve the bed adhesion and reduce the risk of warping or print failure.

Additionally, the choice of materials was restricted to those compatible with FDM, which narrowed the range of mechanical properties available for prototyping. Therefore, reinforcement fins were designed to strengthen the parts.

7.2.6 Material Selection

The development of the enclosure focused on creating a functional and comfortable enclosure housing that could be integrated into the brace. The enclosure comprised of two primary components: a top shell and a bottom plate.

To identify the most suitable material pairing, each component was 3D printed using three different materials: PLA+ (Polylactic Acid composite with impact modifiers), PETG (Polyethylene Terephthalate Glycol) and TPU 95A (Thermoplastic Polyurethane). This resulted in a total of nine possible configurations. After systematic testing, PLA+ was excluded from consideration, given that it showed less plastic deformation ability, shattering upon exerted stress. Therefore, the test candidates narrowed down to 4 different configurations.

A combination of PETG for the top shell and TPU 95A for the bottom plate were selected. PETG offered the ideal balance between rigidity and resilience, providing structural integrity while still allowing for a degree of plastic deformation without fracturing. TPU 95A was chosen for the bottom part for its softness and flexibility as well as its hypoallergenic qualities as a Thermoplastic Elastomer. It offered both user comfort and durability making it the optimal material for prolonged skin contact.

8 INTEGRATION

8.1 ML MODEL INTEGRATION

8.1.1 Hardware Integration

For the machine learning model to correctly classify heart sounds in real-world deployment, it was important that the training data fed into the model matched as close as possible to the output data recorded by the device. Certain features could be decided at the start of the project as part of initial system design (e.g. file format, dataset selection, probable resolution). Other features were more complex, and were decided upon in later stages as both machine learning and hardware tests were carried out (e.g. file duration, sampling rate, filtering effects).

8.1.1.1 File Format

All the model data was formatted as .wav files, selected because it ensured the three selected PCG datasets could be combined with each other without any need to convert files. It was decided the device should output a .wav file to minimise the amount of pre-processing any audio recording would have to go through to match the model. With this format, any adjustments to filtering, sample rate or duration could be done with the same code as applied to the different training datasets. It also allowed for sampling rates to be adjusted when loaded using Python librosa, which allowed more flexibility with what rates the hardware could accommodate.

8.1.1.2 File Duration

As described earlier, all model training files were extended or trimmed to 45s, with abnormal files exceeding 45s being discarded. To balance timing between sampling and filtering/processing, the microcontroller recorded for 10s at a time – therefore the same code used to repeat and extend normal training files could be used to take the 10s recordings, extend to 45s, and then perform the same spectrogram generation. This also allowed the model to have some flexibility if the 10s duration was changed for any reason.

8.1.1.3 Sampling Rate

Sampling rate refers to the number of samples taken per second from a continuous signal. It is generally decided by the Nyquist Theorem [51], in which the sampling rate should be at least twice the frequency of its highest frequency component to reproduce the original signal without losing information or causing aliasing.

$$f_s = 2f_n \tag{5.1}$$

Relevant heart sounds occur between 25 - 500 Hz [47], so for this application the minimum sampling rate was 1000 Hz. To conserve memory and extend the file duration as much as possible, the target sampling rate for the device was 1000 Hz; however, the model training data was sampled at a range of frequencies from 2000 – 4000 Hz across the datasets. Increasing the device sampling rate to 2000 Hz to match most training data would decrease file duration to 5s; it was therefore important to decide which was more relevant for the model. Grid validation was carried out in which all model dataset files were up-sampled to 5000, 10,000 and 22000 Hz, or down-sampled to 2000 and 1000 Hz. Separate CNNs were trained on each frequency. The results demonstrated that increasing or decreasing the sampling rate had very little effect on model learning rates or performance – if needed, the model preprocessing could adjust sampling rates with librosa, regardless of original rate. This again allowed for more flexibility in deciding sampling rate and duration with respect to the microcontroller.

8.1.1.4 Filtering

The main filtering was intended to be carried out on the microcontroller through applying an ATF to remove the acoustic effects of the chest cavity, then adaptive filtering to remove background noise and deliver a clean heart sound. Ideally, the ATF and any fixed filtering could also be applied to the model dataset files to ensure a good match between datasets and real recordings. This was identified as an integration risk; to mitigate it, simpler filtering was developed in the preprocessing for the model dataset files as a 'failsafe'. Therefore, a significant part of the grid validation performed at this stage was comparing the effect of different preprocessing filters on model performance and accuracy.

8.1.1.5 Spectrograms

The final feature considered in model-hardware integration was the type of spectrogram the model was trained on. From literature documenting similar heart abnormality classification problems, the two spectrogram types with the generally highest accuracy rates were Mel-frequency spectrograms and Mel-frequency cepstral coefficient (MFCCs) spectrograms. Which of the two was most successful varied depending on the model and specific use case. To try and account for as much flexibility as possible in the hardware integration, all possible sampling rates and filter preprocessing were repeated in the grid validation for both Mel and MFCC spectrograms.

8.1.1.6 Grid Validation

The purpose of the grid validation was to identify what combination of input file features achieved the highest performance when applied to all model dataset files and put through the base iteration of the CNN architecture. For control, the same split in the original audio directory was used to make up the training, validation and testing set splits for each model – i.e. the test set for the model of 1 kHz Mels with Filter B was the same as for the model of 22 kHz MFCCs with Filter C.

Original referred to the audio file plotted as a spectrogram without any filtering, so often included noise. Filter A was a basic 20-500 Hz bandpass with no other modifications. Filter B was a lowpass built on coefficients from an early ATF derivation. Filter C was a 25-500 Hz fourth-order bandpass with a pre-emphasis filter.

In total, 40 different models were compiled, run, and evaluated as part of the grid validation. The accuracy of each on unseen test data is shown below in Table 12.

Table 12: Grid Validation #1 - Input File Types

	Original		**Filter A**		**Filter B**		**Filter C**	
Mels	1 kHz	53	1 kHz	**82**	1 kHz	71	1 kHz	55
	2 kHz	67	2 kHz	**94**	2 kHz	61	2 kHz	40
	5 kHz	48	5 kHz	**94**	5 kHz	**90**	5 kHz	**95**
	10 kHz	62	10 kHz	43	10 kHz	55	10 kHz	**95**
	22 kHz	56	22 kHz	81	22 kHz	74	22 kHz	80
MFCCs	1 kHz	41	1 kHz	75	1 kHz	**81**	1 kHz	72
	2 kHz	66	2 kHz	71	2 kHz	**96**	2 kHz	65
	5 kHz	80	5 kHz	71	5 kHz	79	5 kHz	81
	10 kHz	**95**	10 kHz	**93**	10 kHz	80	10 kHz	47
	22 kHz	74	22 kHz	80	22 kHz	68	22 kHz	80

From Table 12, the results of the top ten best performing models were plotted as loss curves, to identify any case where overfitting had occurred. The best performing five at this stage were then saved to be compared and narrowed down again in integration testing – when device recordings could be used in the test set to better direct architecture parameter tuning.

8.1.2 Application Interface

8.1.2.1 Model to Android Studio

One of the key advantages of using TensorFlow to develop the ML model was its compatibility with TensorFlow Lite – a Google framework optimised for running machine learning on devices such as microcontrollers, mobile applications, and embedded systems [52]. It allows models to be compressed to .tflite files which can then be run directly on the target device, optimising for resource constraints and low power consumption. In addition, Android Studio includes dedicated support for deploying TensorFlow Lite models in every version after v4.1.

To integrate the ML model with the application, the model was compiled and trained using Keras, TensorFlow's high level API. It was saved in standard keras format as a back-up, then converted into TFLite SavedModel format using TensorFlow's TFLiteConverter. This saved model could then be written to a .tflite file and exported.

```
Model.save('CNN_R6_BPA_Mels.keras')
converter = tf.lite.TFLiteConverter.from_keras_model(Model)
tflite_model = converter.convert()

with open("CNN_R6_BPA_Mels.keras", "wb") as file:
     file.write(tflite_model)
```

After the model was exported as a .tflite file it needed to be imported to Android Studio. This was done by copying it into the 'assets' folder and adding the relevant dependencies [53], [54]. To interact with the Model, it had to be represented as an Object in the code, it was imported as a 'org.tensorflow.lite.support.Model' [55]:

```
    Model model = Model.createModel(context, "<file_name>.tflite");
```

8.1.2.2 Preprocessing

Before the model could be run, the audio file had to be converted to the type of input it was expecting. This was done in two stages, creating a spectrogram (in Python) and converting that to a ByteBuffer with relevant format (in Java).

8.1.2.3 Running Python from Android App (Chaquopy)

Python has several libraries and methods for graphs, plotting, etc. and most do not have Java equivalents. Therefore, it was decided that the simplest way to perform the initial pre-processing was to run Python code from the Java app. In a computer setting this would be relatively simple, most likely creating a .exe file to be run or a script. However, this cannot be done on a mobile phone, so another tool had to be used.

Chaquopy is a tool that can be used to run Python on a variety of platforms, in this project the Gradle plugin was used to integrate with Android Studio's build platform. The steps in [56] were followed to add the Chaquopy plugin:

- Add "id("com.chaquo.python") *version* "16.0.0" *apply* false" to the top-level build.gradle file
- Add "abiFilters += *listOf*("arm64-v8a", "x86_64", "x86")" to the android defaultConfig ndk section of the app-level build.gradle file
- Add a Chaquopy block in the app-level build.gradle file:

```
chaquopy {
    defaultConfig { }
    productFlavors { }
    sourceSets { }
}
```

When using libraries in Python, they have to be installed using Python's installer: pip. Similarly, a pip block is added within the defaultConfig of the Chaquopy to indicate which libraries need to be included. It is vital that versions of each of the libraries and the version of Chaquopy are all compatible with each other. In this case, the most recent version of one of the libraries (librosa) was not compatible with Chaquopy, resulting in build failures. This was fixed by specifying an earlier version of librosa and earlier versions of some of the libraries it is dependent on [57]:

```
pip {
    install("librosa==0.9.2")
    install("resampy==0.3.1")
    install("matplotlib")
}
```

Once the tool was setup the python file with code to be run was added to the 'app/src/main/python' directory. Finally, an instance of Python could be created and run from the Java Android Studio code:

```
if (!Python.isStarted()) {
    Python.start(new AndroidPlatform(context));
    py = Python.getInstance();
}
```

A copy of the Python pre-processing code can be found at Appendix C. Everything is run via the 'main' method so that only one call needs to be made. This includes creating a waveform image to be used to display to the user on request as this is significantly easier in Python. This does have the drawback of requiring the images to always be stored on the phone rather than generated when they are needed. This takes up additional storage space, however they can be deleted by the user without affecting the rest of the recording/application. To run the code, the following command is used:

```
pyDataProcessing.callAttr("main", <targetLength>, <input_file_path>,
<path_for_temp_extended_file>, <path_for_temp_spectrogram_file>,
<path_to_store_waveform_image>);
```

Where 'main' is the function to be called and everything following are its parameters. The result of this is a spectrogram of the input .wav file saved on the phone at the path given. This is then used for further pre-preprocessing in Java.

8.1.2.4 Pre-Processing Spectrogram into Correct Format

The model takes the images in as a 3D array of the form:

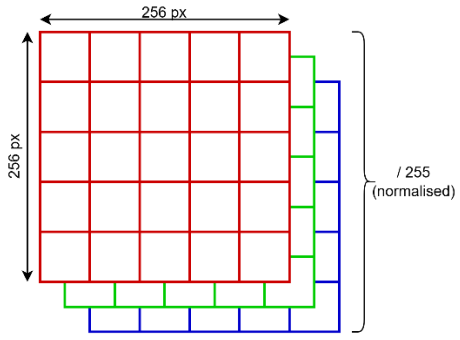


Figure 42. Model input form diagram

The steps to convert the spectrogram are as follows:

- Scale to 256 x 256 pixels
- Create a ByteBuffer to store the results (a byte array wrapped in an Object to facilitate interaction)
- Loop through each pixel in the image:

- Get red, green, and blue values
- Normalise (divide by 255)
- Add to ByteBuffer

8.1.2.5 Running the Model

The model is capable of processing multiple inputs in one batch, so it expects an array of Objects as an input and returns a Map of Integers (batch number) to Objects (the actual output) as the result. Only one input is ever processed at once in this case, so an array of one Object (the ByteBuffer) is passed in and a Map containing one encoding is given for the result:

```
float[][] output = new float[1][2];
Map<Integer, Object> out = new HashMap<>();
out.put(0, output);

model.run(new Object[]{input}, out);
```

8.1.2.6 Postprocessing

The output is a 2D array of the form [[normal probability], [abnormal probability]], where each probability is a decimal value between 0 and 1 with the model's likelihood of it being that classification.

The integer corresponding to the classification of the highest probability is returned (0 – abnormal, 1 – normal). However, if the probabilities are within 0.25 of each other – i.e. 37.5 <= probability <= 62.5 - then the integer corresponding to noise/unknown (2) is returned.

Finally, the integers are interpreted, and a list of Strings is returned to the main program to add to the relevant recording by calling (as discussed in Section 5.5.1.10):

```
callback.onComplete(tagsToAdd);
```

Any temporary files generated, i.e. the extended .wav file and the spectrogram, are deleted prior to returning.

8.1.3 Human Interface

8.1.3.1 Auscultation Point

There are five areas on the human chest that make up the primary cardiac auscultation points: aortic, pulmonic, Erb's, tricuspid, and mitral. Each area influences detection of the main heart sounds S1 and S2. Generally, the clearest S1 sound can be heard at the mitral point and the clearest S2 at pulmonic or Erb's, with tricuspid or aortic as a balance between the two. With only one microphone in the device, a single auscultation point had to be selected. For proper model integration, the auscultation point should match where the dataset files were recorded, but none of the three datasets provided detail on auscultation location.

8.1.3.2 CirCor Dataset

The CirCor DigiScope Dataset was identified in early design stages but was not used as a training dataset due to it being comprised of largely paediatric heart sounds and classified as such – unsuitable for the project as a heart sound normal for a child could be abnormal for an adult, and the CirCor dataset did not clarify these cases. It contains 5282 recordings from 1568 patients, with each recording labelled by auscultation point: pulmonic (PV), aortic (AV), tricuspid (TV) and mitral (MV). Together, MV and TV form the S1 sound. AV and PV form the S2 sound. To determine if there was a difference on model performance depending on auscultation point, the CirCor dataset was split into two subsets by either S1 or S2 bias, and then a CNN trained on each subset.

8.1.3.3 Model Comparison

Both models were built with the base CNN architecture detailed in Section 6.1.1, compiled with the same dataset split, and given the classification problem of 'healthy' or 'unhealthy'. The model trained only on S1 biased sounds achieved an accuracy of 75% on the training set, 62% on the validation set, and 60% on the test set. The model trained only on S2 biased sounds achieved an accuracy of 66% on the training set, 58% on the validation set, and 55% on the test set.

Overall, there was only minor performance differences between the S1 and S2 models. When looking into more detailed classifications – i.e. how many systolic and diastolic abnormalities each model classed as normal/abnormal – there was a decrease in how effective the S2 model was in detecting systolic abnormalities, while the S1 model performed consistently for both systolic and diastolic. Given that most heart murmurs occur in the systolic region, it was concluded that a clear S1 sound was more important for a better overall signal. To get the best S1 sound possible, the mitral auscultation point was selected.

8.2 APPLICATION WITH HARDWARE

To allow the application to communicate with the device, the application implements Bluetooth communication. The class responsible for communicating with the device was the Device Controller class.

8.2.1 Permissions

Before any Bluetooth communication can commence, the app must acquire permissions from the phone. In this case the application needed access to the Bluetooth capabilities and its fine location which are used together when scanning for nearby devices. The exact permissions required to access Bluetooth on the device vary from manufacturer to manufacturer and also based on Android version.

For older versions of Android (11 and below), only require a Bluetooth permission and a Bluetooth admin permission. This has changed in android 12 where the permissions are required for more precise actions instead, in this case Bluetooth scan permission and Bluetooth connect permission.

As mentioned above, these permissions also vary device manufacturer to device manufacturer where some follow the procedure outlined in the Android development documentation while some have their own approach such as granting applications automatic access to Bluetooth services and the above-mentioned permissions not existing on the device.

The code design accounted for majority of these edge cases by requesting different permission from the device based on the current Android version. The rest of the description will focus on the primary development device used in this project: a Xiaomi phone running Android 15.

When the application is launched for the first time, the user is asked to give the application the permissions it requires. The Bluetooth and fine location permissions are presented in two separate popup messages. If the user does not grant the permissions the application will not be able to function as intended.

8.2.2 Connection

Once the user grants the required permissions, the device controller begins a Bluetooth Low Energy scan of nearby devices. The Bluetooth scan has a timeout window of 10 seconds as a safety measure due to Bluetooth scanning's high power consumption. The Bluetooth scan stops in two cases: the 10 second time out window is met or it manages to find a device with the name "blehr_sensor_1.0". When the device is found, an instruction is sent to establish a connection with the device. Upon successfully establishing the connection, the application initiates a Bluetooth callback informing the device of a change in the Bluetooth state. Any listeners who are subscribed to the Device Controller are notified of

the new state (Connected in this case). The application then proceeds to request a list of offered services from the newly connected device. Once the list of services is received, the application begins to set parameters of certain characteristics and subscribe to others.

8.2.3 Main Process

The application begins by getting the current time and writing the value to the time update characteristic to inform the device of the current UCT time. Next the application subscribes to the OTS and informs the phone to expect incoming indication messages. At this point the device is fully connected to the application with no further setup necessary and is ready start receiving recorded heart sounds as long when the mode is changed from standby by the application.

To begin recording, the application updates the mode characteristic value to continuous. 10 seconds later it will receive the first data packet. Since this is the first packet and there aren't any ongoing data transfers, it treats the packet as file name information and sets a flag that file transfer has begun. Any consequent data received is placed inside of a byte array until it receives a packet of size 1, at which point the data transfer for that specific file has finished. The byte array containing the contents of the file and the file name and its extension are then passed onto the File Handler to save the file. At the same time, Device Controller observers are updated of a Bluetooth state change (NEW_DATA) which starts the process of calling the ML model to analyse the recording. Finally, the data transfer flag is reset. The application is now ready to receive new data.

Whenever the device is disconnected, expectedly by the app itself or unexpectedly, the Device Controller updates its observers of the change in Bluetooth state.

9 TESTING

9.1 TEST PLAN

There is no established procedure for evaluating auscultation devices. Additionally, testing the overall function on human subjects (aside from group members) was not feasible due to ethical concerns and insufficient testing populations. As such, the assessment of this device relied on alternative approaches guided by existing literature on similar technologies. The aim was to ensure that these tests demonstrated the system's ability to function effectively in conjunction with the human body under a wide range of conditions.

Additionally, standard audio properties such as Sound to Noise Ratio (SNR), Total Harmonic Distortion (THD), Acoustic Overload Point (AOP), noise floor and frequency response will be evaluated to characterise it as a standalone audio device.

An acoustic phantom was constructed to validate the human-device interface under realistic operating conditions. The phantom consists of a sound source and a material with similar acoustic properties to tissue, simulating how sound propagates through the chest wall. For this project, a synthetic gel (ECO-FLEX Gel 2) that is often used in medical environments was chosen as the tissue-mimicking material. and other apparatus such as a piezoelectric buzzer, an enclosure and acoustic foam were sourced. Table 13 (following page) outlines the various tests that assess device performance in simulated and real-world environments with realistic operating conditions such as noise and daily movement.

A continuous wearable heart sound sensor should be able to record suitable heart sounds from a range of people performing normal daily activities (e.g. talking, walking, jogging, exercising) and in realistic noise conditions (ranging from quiet lab conditions to chatter).

Table 13. Summary of test plan

Test Objective	Testing Subject	Testing Methodology	Results Interpretation
Can it record a heart sound from a person?	Group Members	Use device on group members (quiet environment, resting heartbeat)	Comparison of recorded signal to expected signal (from data sets or other heart measurement device)
Can it identify abnormal heart sounds?	Acoustic Phantom	Play pathological and healthy heart sounds on acoustic phantom (quiet environment)	Comparison of heart sound classification to actual heart sound classification
Can it record suitable heart sounds in noisy environment?	Group Members	Use device on group members (noisy environment)	Qualitative assessment of signal
To what extent can it identify abnormal heart sounds in noisy environments?	Acoustic Phantom	Play pathological and healthy heart sounds on acoustic phantom while playing noise samples at various volumes in a controlled environment	Comparison of heart sound classification to actual heart sound classification
Can it record suitable heart sounds while user is performing common activities?	Group Members	Use device on group members while they are standing still, walking, jogging, talking	Qualitative assessment of signal, checking for motion artifacts
To what extent does device positioning matter in recording suitable heart sounds?	Acoustic Phantom	Place speaker in different positions underneath tissue-mimicking material	Quantitative comparison of reference sound and recorded sound at different distances from source

9.2 ACOUSTIC PHANTOM

To simulate the acoustic environment the device would encounter, an acoustic phantom was developed to mimic the characteristics of the human chest cavity. The phantom included a piezoelectric buzzer as a sound source element and was designed with multiple layers of material, selected for their acoustic and vibrational properties. The housing directly beneath the buzzer was 3D printed in a PETG plastic as a shell with three walls, three top and bottom layers, from a 0.4 mm nozzle, 0.2 mm layer height and 15% gyroid infill, with cavities to fill in with polyurethane foam. This structure laid on top of a custom compound consisting of 50 parts ‘Plaster of Paris’, 37.5 parts water and 27.5 parts Polyvinyl Acetate (PVA) glue enclosed in a 3D printed TPU 95A shell with three walls, three top and bottom layers, from a 0.4 mm nozzle, 0.2 mm layer height and 15% gyroid infill. This was made in an effort to absorb excess vibrational energy and minimise internal reflections, thus reducing “echo-chamber” effects and allowed for a more controlled, directional acoustic transmission.

An alternative Plaster of Paris mix – 50 parts plaster, 25 parts water and 7.5 parts PVA glue - was also composed to be evaluated inside a 3D printed PETG shell with three walls, three top and bottom layers, from a 0.4 mm nozzle, 0.2 mm layer height and 15% gyroid infill. However, this configuration performed poorly, and mechanical issues during the de-moulding process prevented a direct comparison across all material and shell combinations.

The electronic configuration of the phantom included a 12V power supply, a VHM-314 Bluetooth 5.0 DAC and a TPA3116 amplifier which replaced the initially used Fosi Audio BT20A Pro for cost-effectiveness. A 12V-to-5V buck converter was used to power the DAC. A range of resistors (10 MΩ to 4.7 kΩ) were tested iteratively to fine-tune the output characteristics, placed across the buzzer to lower the total resistance and increase current flow, effectively boosting output volume. The optimal resistor value was found to be 4.7 kΩ.

The DAC, buck converter and power supply were deliberately left external to the main housing to allow for modularity, control and easy adjustments during testing.

The phantom was calibrated using a high-accuracy microphone from a mobile phone to verify that the tones played back, corresponded to what was input as a signal - impulses at certain frequencies within the frequency region of interest.

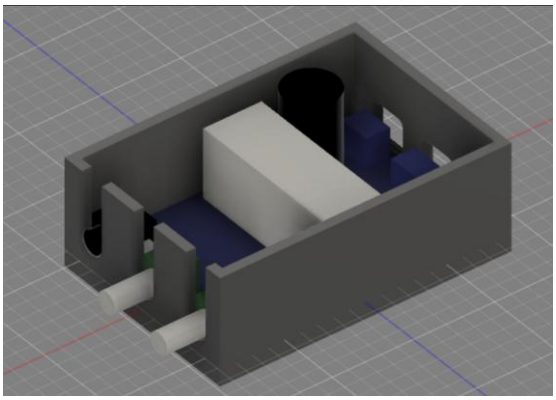


Figure 43. Amplifier case.

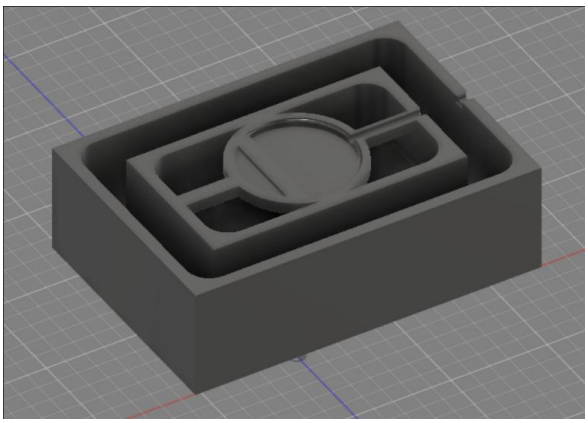


Figure 44. Piezoelectric buzzer casing

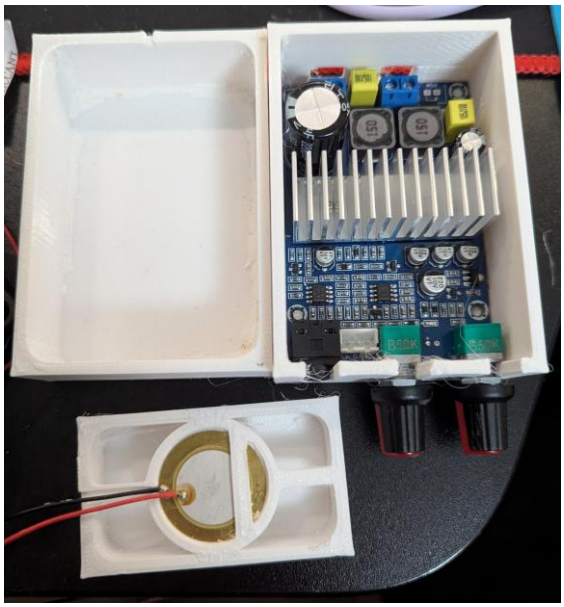


Figure 45. Acoustic Phantom

9.3 MATERIAL TESTING

To replicate the acoustic transmission characteristics of human skin, particularly over the mitral point, a series of candidate materials were tested and evaluated. The aim was to identify a material that most accurately reproduces the attenuation, delay and spectral filtering properties observed in actual human skin when recording PCG signals.

To achieve this, the acoustic phantom was used to simulate heartbeats and play impulses at defined frequencies within the cardiac frequency band of interest (20-500 Hz). The testing involved placing a series of materials directly on the phantom and recording acoustic responses both with and without material interference. This allowed to isolate and quantify the transmission loss (TL) and filtering properties introduced by each material.

9.3.1 Experimental setup and material selection

Initially, the project plan relied on materials that were readily available in the lab environment, including Kaizen foam and polyurethane-based packaging inserts. These were selected for their general acoustic damping properties and ease of handling. However, preliminary testing revealed that these materials did not provide transmission characteristics close to that of human skin, particularly in the frequency range relevant to heart sounds. In response to this, a more targeted approach was adopted: EcoFlex Gel2 [58], a soft silicone gel commonly used in Hollywood for skin effects, was sourced to fabricate custom skin surrogates.

To facilitate this, two moulds were designed and 3D printed using PLA+ filament, tailored specifically to create gel pads of uniform thickness (10 mm and 20 mm). The EcoFlex Gel2 was then mixed, poured into the moulds, and allowed to cure under controlled conditions, according to manufacturer's instructions.

This adjustment to the experimental pipeline significantly improved material reproducibility and allowed for better control of geometric and acoustic parameters during testing.

9.3.2 Signal processing and Transmission Loss estimation

To evaluate the acoustic filtering properties of each material, transmission loss was computed using the magnitude spectrum of the Fast Fourier Transform (FFT) as:

$$\mathrm{TL(f)} = 20\,log_{10}\left(\frac{|\mathrm{FFT(No\ material - noise)}|}{|\mathrm{FFT(With\ material - noise)}| + \varepsilon}\right) \tag{9.1}$$

Where ε is a small constant to avoid division by zero.

All signals were highpass filtered at 20Hz to remove environmental and structural noise. Where necessary, Hann windowing was applied prior to transformation. These TL curves provided a frequency-dependent attenuation profile for each material. In addition to the TL estimation, Welch's Power Spectral Density (PSD) method was used to produce smoothed, normalised plots for qualitative comparison.

Plots were grouped into foam-based and gel-based materials to preserve clarity and interpretability. Overlaying all materials in a single plot was initially attempted but proved visually congested due to the density of overlapping spectra. These are shown in Figure 46 and Figure 47.

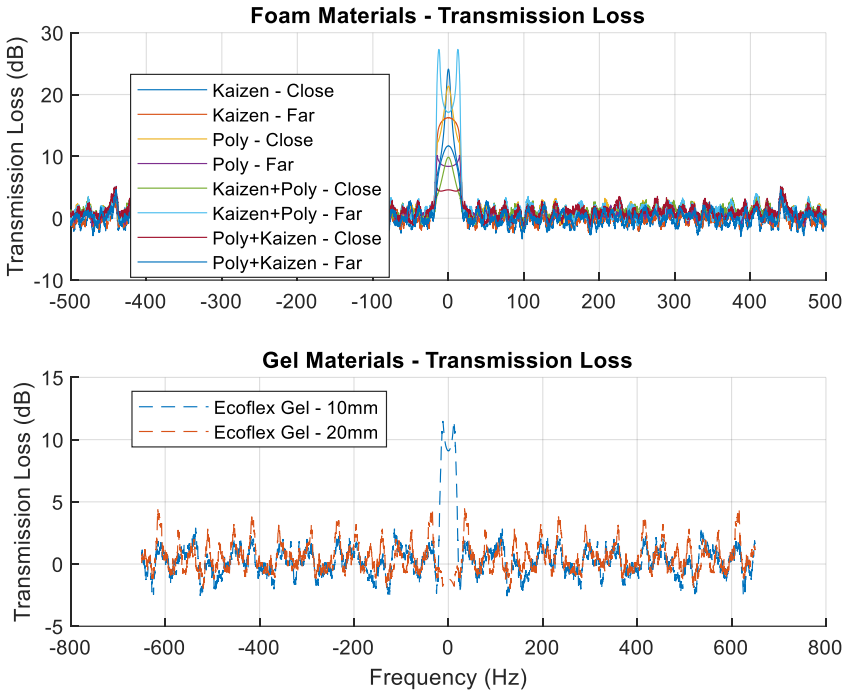


Figure 46. Transmission loss for all the materials

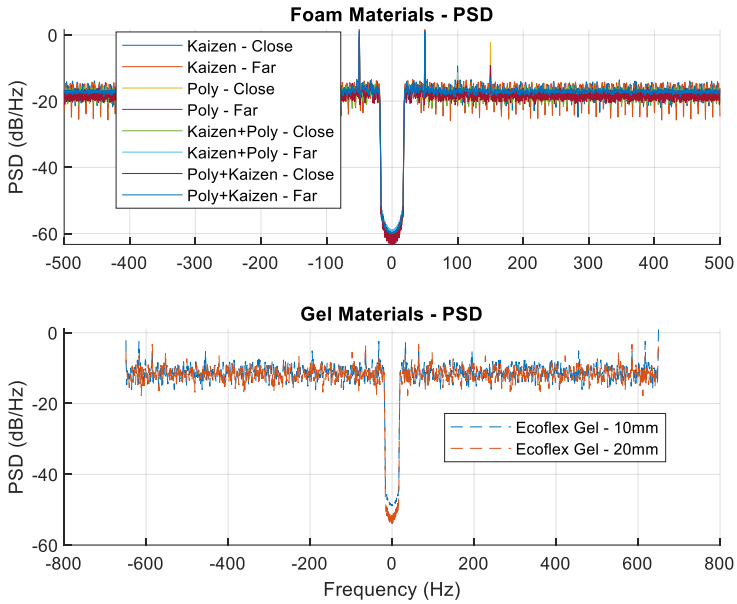


Figure 47. PSD plots for all the materials

9.3.3 Inverse filter derivation and validation

To assess how well each material emulates the skin's acoustic behaviour, inverse filters were derived in the frequency domain using the method of spectral division. Specifically, a clean reference signal was divided by the recorded output signal for the materials. The resulting transfer function was then inverted and transformed back into the time domain using the inverse FFT. A Hann window was applied to suppress edge effects, and the impulse responses were truncated to 64 and 512 taps for use as FIR filters.

This approach enabled direct comparison between the acoustic effect of each material and a skin transfer function derived from the ECG-PCG signal analysis, as described in the next section.

To validate the performance of each inverse filter, its response was compared to the derived skin transfer function obtained from ECG-PCG signals as discussed in 4.2.3 and the MSE between this inverse filter and the inverse filter derived from each material was computed as:

$$\mathrm{MSE} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\left(\mathrm{h}_{\mathrm{inv,material}}[\mathrm{i}] - \mathrm{h}_{\mathrm{inv,ecg-pcg}}[\mathrm{i}]\right)^2 \tag{9.2}$$

With MSE values summarised in

Table 14. MSE values for different materials.

Material	MSE with ecg-pcg
EcoFlex 20 mm	8.2×10^{-4}
EcoFlex 10 mm	1.6×10^{-3}
Kaizen + Poly	7.5×10^{-3}
Polyurethane	9.1×10^{-3}
Kaizen	1.1×10^{-2}

The EcoFlex Gel2 20 mm exhibited the lowest error, indicating it is the most acoustically precise surrogate for simulating the skin over the mitral point. Time and frequency domain plots, shown in Figure 48 and Figure 49 respectively, confirmed that the inverse filtering effectively restored high-frequency content attenuated by the skin cavity, offering positive information gain.

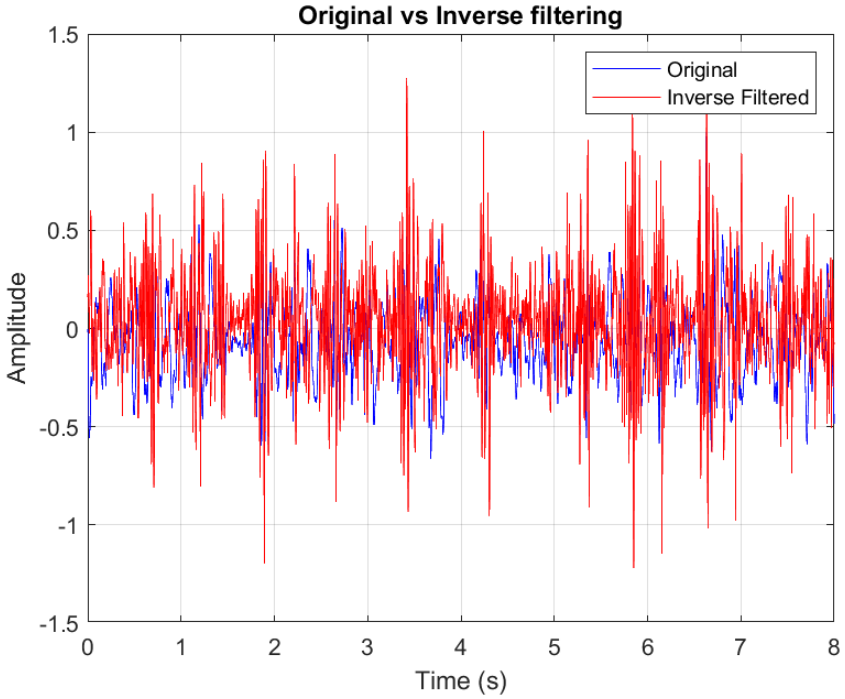


Figure 48, Original vs Inverse filtered in time domain

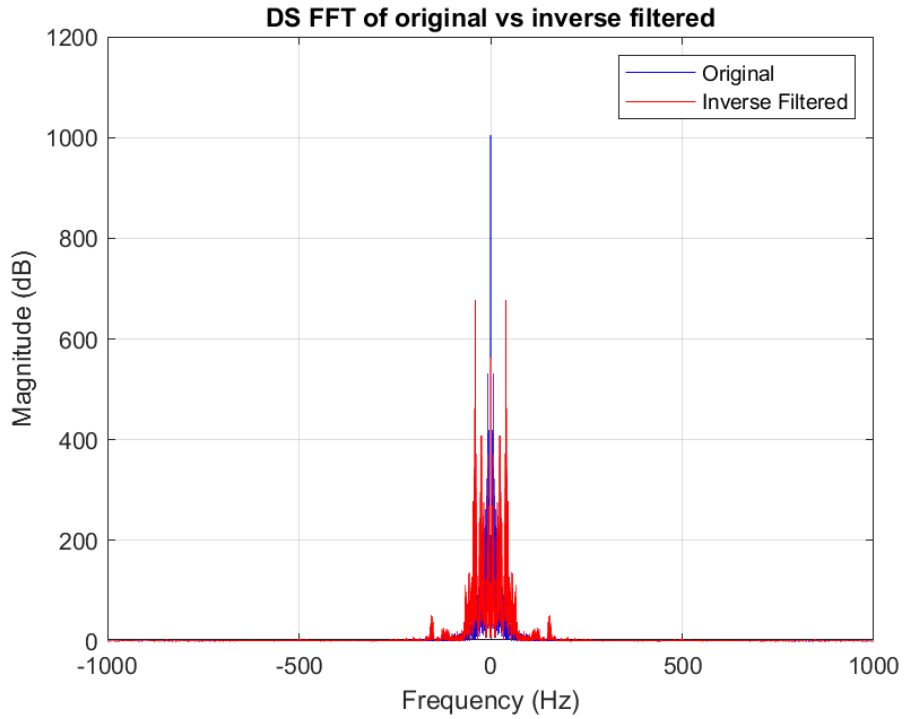


Figure 49. Original vs inverse filtered in the frequency domain

9.3.4 Implementation for embedded

In addition to numerical evaluation, the best-performing inverse filter (EcoFlex Gel2 – 20 mm) was prepared for use in the MCU filtering – the 64-tap FIR filter coefficients were quantised and exported as a C header file for the MCU (section 4.3.3).

9.3.5 Observations

Through comparative spectral and temporal analysis and validation with inverse filter testing, it was concluded that the EcoFlex Gel2 at 20 mm thickness was the most suitable candidate for simulating the skin over the mitral point. It preserves low-frequency content while smoothly attenuating higher frequencies, exhibits a low MSE when matched against a real skin-derived transfer function and performed well in time-domain reconstruction tasks.

Foam-based materials, while effective at attenuation, introduced excessive spectral distortion, particularly at higher frequencies and exhibited inconsistent delay characteristics. These properties make them less suitable for physiological simulation where realistic transmission is desired.

9.4 Acoustic Property Testing on MEMS

MEMS microphone membrane characterisation was carried out within an acoustic booth using a dual-reference configuration. The MEMS microphones were mounted horizontally with a speaker positioned to the right a reference Brüel & Kjær 4138-A-015 1/8” pressure field microphone placed to the left, and a PolyTec MSA-100 3D Laser Doppler Vibrometer (LDV) mounted overhead to measure vibrational velocity directly on the membrane surface. Excitation signals included a 1 kHz pure tone sine and a broadband periodic chirp sweep. The acoustic level was calibrated to approximately -20 dBV SPL, corresponding to a source output of 100 mV/Pa, a common benchmark value for microphone characterisation.

Environmental conditions within the test booth were monitored and recorded. Temperature ranged from 17.9°C to 28.4°C and relative humidity varied between 10 and 60%. At the time of testing, the ambient environment was maintained at 22.4°C and 10% RH, which lies within but toward the edge of the

standard IEC 60268-4 microphone characterisation envelope. The LDV laser operated at a wavelength of 532 nm under low-power continuous wave (CW) operation, with alignment optimised for out-of-plane displacement measurement.

To characterise the acoustic-to-mechanical coupling behaviour of the MEMS microphones, the transfer function was computed as the frequency-dependent ratio of the membrane's vibrational velocity (in m/s) to the reference sound pressure (Pa):

$$H(f) = \frac{v_{mems}(f)}{p_{ref}(f)} \tag{9.3}$$

Both peak and average values of this transfer function were calculated and compared to the expected performance of the MEMS microphones' datasheets.

Figure 50. PolyTec MSA-100 3D LDV

Figure 51. B&K sound pressure microphone

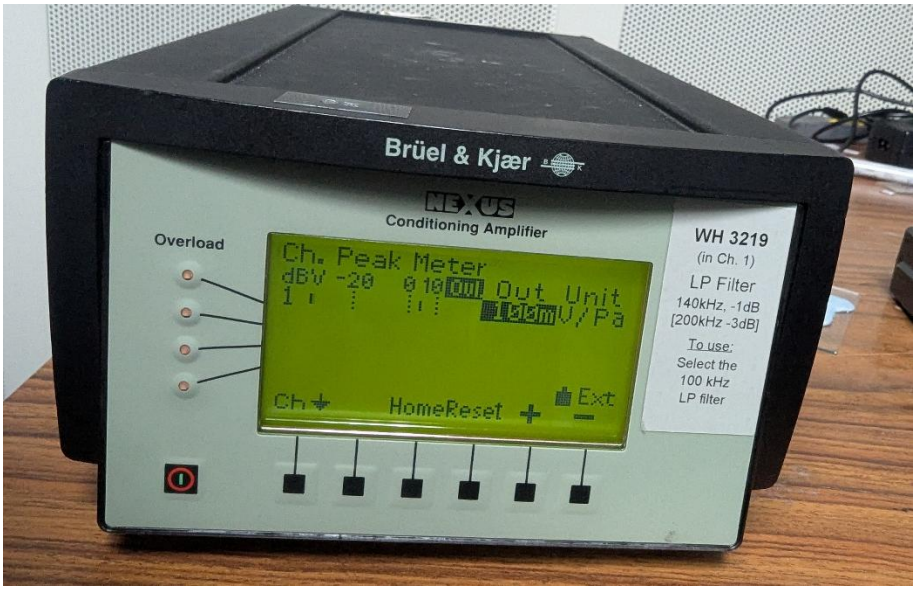


Figure 52. WH3219 Amplifier

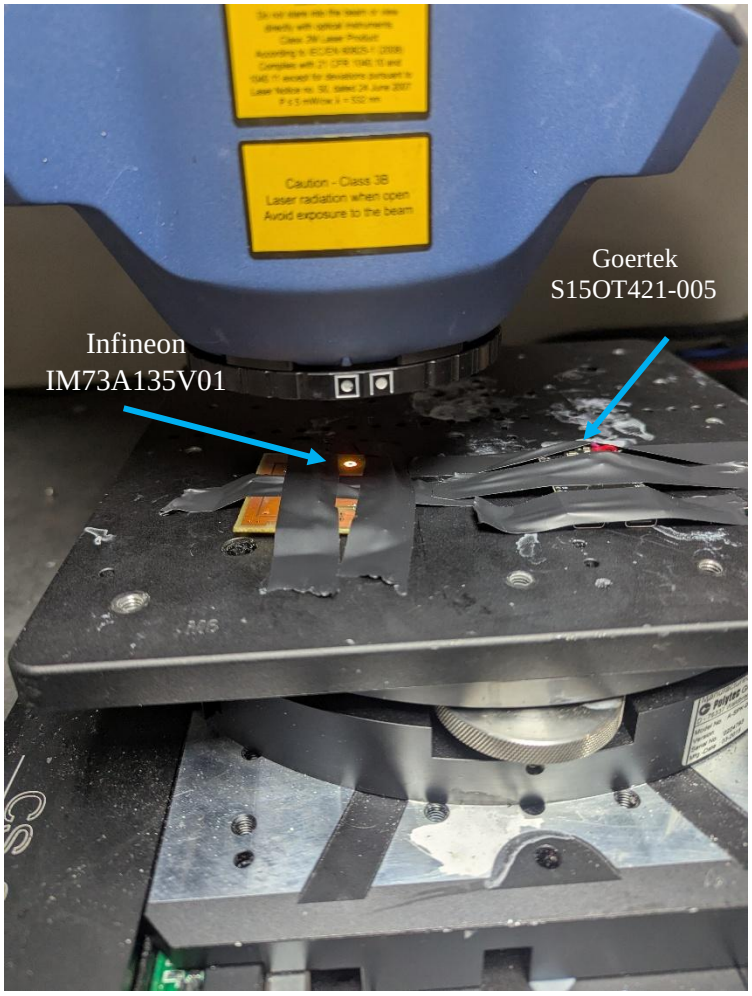


Figure 53. MEMS testing setup

9.4.1 Goertek S15OT421-005

The Goertek S15OT421-005 MEMS microphone is specified with a sensitivity of -42 dBV/Pa at 1 kHz, corresponding to an electrical output of 100 mV RMS for a 1 Pascal acoustic input. Under these same conditions, replicated in the acoustic booth, the measured mechanical response of the microphone membrane, captured via vibrometry, revealed a distinct resonant peak at 1 kHz, confirming the manufacturer's defined sensitivity reference point. The calculated mechanical transfer function, defined as the ration of vibrational velocity to incident pressure, exhibited a peak of -14.93 dB (re: m/s/Pa) at 1 kHz, with an average of -43.2 dB across the bandwidth for the sine tone condition.

During the broadband chirp frequency sweep, the peak transfer function was -29.43 dB, and the average -45.32 dB. These values demonstrate that while the mechanical response sharply resonates at 1 kHz, the overall diaphragm behaviour attenuates more rapidly outside this region than the ± 3 dB response envelope presented in the datasheet.

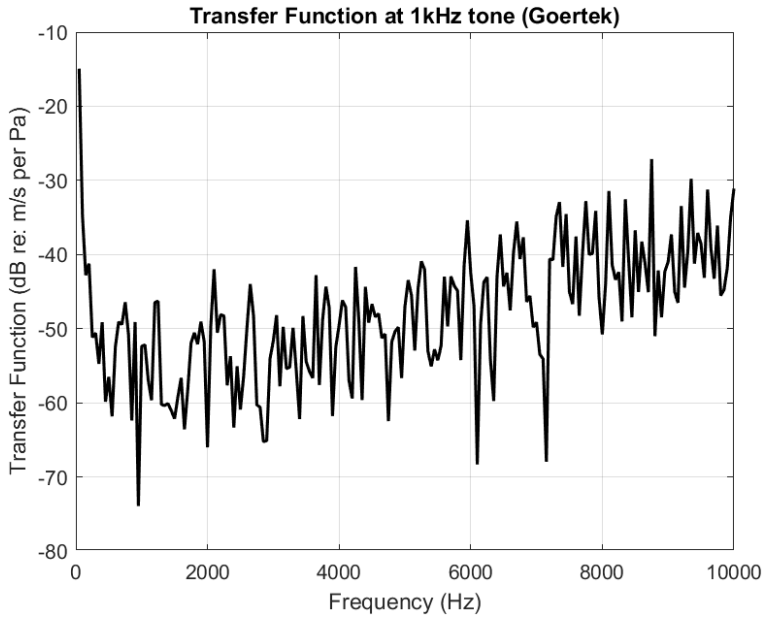


Figure 54. Goertek S15OT421-005 1 kHz sine TF

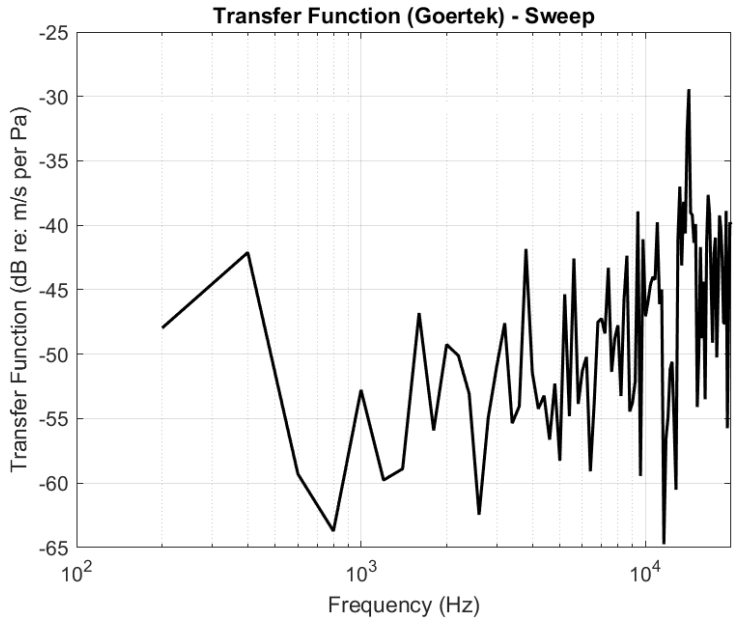


Figure 55. Goertek S15OT421-005 frequency sweep TF with chirp

9.4.2 Infineon IM73A135V01

The Infineon IM73A135V01 MEMS microphone is specified to deliver a sensitivity of -38 dBV/Pa at 1 kHz, equivalent to approximately 126 mV RNS for a 1 Pascal incident pressure wave.

In the 1 kHz sine tone test conducted under identical calibrated conditions, the membrane exhibited a peak vibrational velocity of approximately 2.85 μm/s, corresponding to -110.91dB (re:m/s). This relatively high magnitude when converted to mechanical power per unit pressure, aligns with the expected diaphragm displacement implied by the electrical output level – supporting the integrity of the measured response.

In the broadband chirp frequency sweep test, the calculated mechanical transfer function peak at -9.26 dB (re: m/s/Pa), with an average value of -24.07 dB across the 200 Hz to 10 kHz band. These values indicate that the diaphragm effectively couples acoustic energy into motion across a wide frequency range. The sweep data also showed a consistent roll-off beyond 10 kHz, in agreement with the datasheet [59], which specifies a flat frequency response of ± 1 dB up to 8 kHz and a + 4 dB rise near 10 kHz. While the datasheet's curve describes electrical output, the mechanical transfer function exhibits a shape that reflects similar bandwidth and uniformity, affirming that the diaphragm and backplate geometry are optimised for broad, high-fidelity acoustic performance.

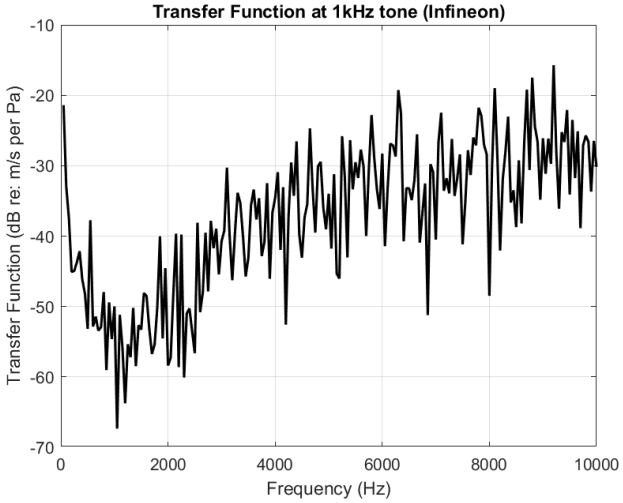


Figure 56. Infineon IM73A135V011 kHz sine TF

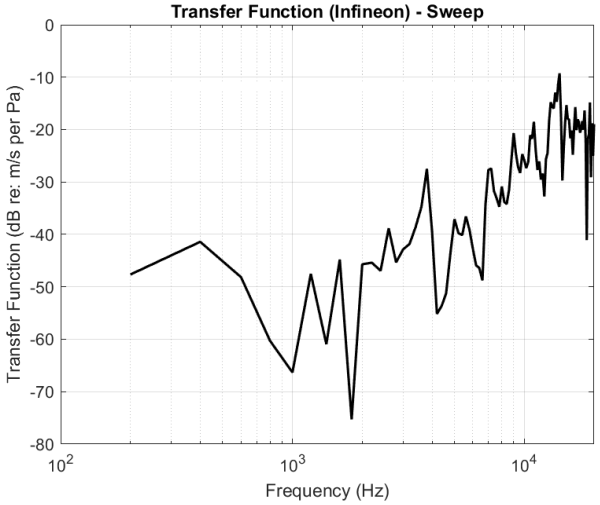


Figure 57. Infineon IM73A135V01 frequency sweep TF with chirp.

9.4.3 Comparison

When compared directly, the Infineon IM73A135V01 exhibited a more uniform and extended frequency response than the Goertek S15OT421-005, with a higher peak and average transfer function across the tested bandwidth. The Goertek MEMS microphone showed a sharper resonance peak at 1 kHz, suggesting a narrower operational band and potentially greater selectivity.

In contrast, the Infineon MEMS microphone, maintained vibrational energy across a broader range, indicative of a diaphragm and housing designed optimised for wideband audio applications. These distinctions are consistent with their intended use cases and datasheet claims and they highlight the value of non-contact mechanical measurement as a complementary tool to the conventional electrical characterisation.

9.4.4 Results

The tested vibrational behaviour of both MEMS microphones under calibrated acoustic excitation aligns closely with their respective datasheet specifications, confirming the validity of the LDV-based measurement method. For the Goertek S15OT421-005, the mechanical response demonstrated a clear resonance at 1 kHz, with strong alignment to its -42 dBV/Pa sensitivity reference. This is attributed to the fact that the datasheet response curves represent post-conditioned electrical output, whereas the LDV captures unfiltered diaphragm motion. Despite this distinction the measured peak and average transfer function support the manufacturer's claimed performance. Environmental variables such as low relative humidity and moderate temperature during testing may gave slightly altered membrane compliance and damping, subtly shaping the dynamic response. Nonetheless, the Goertek diaphragm performed as expected within standard operating conditions. Similarly, the Infineon IM73A135V01 exhibited a broad, high-fidelity spectral response with a pronounced resonance at 1 kHz and minimal roll-off across the sweep range. Its mechanical transfer function closely followed the shape of the datasheet's flat electrical response curve, verifying its wideband sensitivity and acoustic efficiency.

Overall, the testing confirms that both devices function well within their specified operational envelopes when characterised via laser vibrometry.

9.5 INTEGRATION TESTING

Integration testing was vital for this project as there were a significant number of components that all needed to work seamlessly together for the project to be successful. The approach to integration testing was to methodically introduce each part together into the critical path (i.e. the end-to-end process) until the whole system was combined. This approach also allowed for investigation into edge cases and thorough evaluation of each individual component. It should be noted that, for ethical reasons, all tests requiring a human subject were performed on group members.

9.5.1 Hardware Integration

9.5.1.1 Electronics with Physical Housings

The objective of these tests was to ensure that the PCB, battery, enclosure and brace all fit together comfortably on a human end user. To narrow down improvements to specific components at a time, integration testing was kept as modular as possible; therefore, these tests were performed without any of the powered elements of the system. This included delivering power to the PCB, uploading code to the MCU, or running the application.

Testing was carried out as an iterative process, in which tests were completed, improvements were made to each component, and then that component was tested again under the same criteria. This was repeated until the results were satisfactory. These results impacted the iterative processes described in Sections 4.4, 7.1, and 7.2. The findings from the first iteration are shown in Table 15 (following page).

Table 15. Hardware Integration Testing: Electronics with Physical Housings - 1st Iteration

Iteration 1			
Test #	Components	Positive	Improvements
1	Enclosure w/PCB	Enclosure backplate and PCB fit together tight	MEMS cut-out in the backplate doesn't exactly align with the MEMS membrane in the PCB
		MEMS is flush to body	
		Elastic clip fits onto enclosure	Improve tolerances so screw tubes fit over threaded inserts
2	Above (#1) in Brace	-	Clasp holds elastic too tightly
			2 battery PCBs make enclosure too tall, switch to 1 PCB with batteries mounted on both top and bottom side
3	Above (#2) on person	Stays in place well during twisting motion	Elastic ends trail, add loop to tuck in
		Fairly comfortable and intuitive to adjust	Brace shifts a lot up and down, add shoulder strap
			Enclosure moves within brace, make fabric pouch tighter

The main improvements made after the first iteration concerned tolerances within the 3D printing used for the enclosure, finding a suitable balance between comfort and size/fit, and keeping the device in place during human movement. Concurrent hardware tests (as detailed later in this section) also required some adjustments.

These changes were made, and the tests repeated. The findings from the second iteration are shown below in Table 16 (following page).

The main improvements made after the second iteration were to perfect the fit of all finalised components and improve overall user comfort and experience. The findings from the third and final iteration are shown in Table 17 (following page).

Table 16: Hardware Integration Testing: Electronics with Physical Housings - 2nd Iteration

Iteration 2			
Test #	Components	Positive	Improvements
4	Enclosure w/PCB	New battery fits into redesigned enclosure with PCB*	Need to trim outer resin from PCB to fit tight Widen tolerances slightly for perfect fit Add a switch accessible from outside of enclosure
5	Above (#4) in Brace	Elastic clasp removed Shorter enclosure fits much better	-
6	Above (#5) on person	Elastic ends neatly tucked away	Bottom elastic twisting and crumping due to additional pull from shoulder strap, add foam stabiliser Enclosure is now longer, leaves the two horizontal bands too close together

*Between Iteration 1 and 2, concurrent electronics tests revealed a different battery would be required

Table 17: Hardware Integration Testing: Electronics with Physical Housings - 3rd Iteration

Iteration 3			
Test #	Components	Positive	Improvements
7	Enclosure w/PCB	Everything fits well together, including new power switch Everything stable even when shook/under stress conditions	-
8	Above (#7) in Brace	-	-
9	Above (#8) on person	Fit is comfortable and easy to adjust without help Stays secure even during movement	-

After this iteration, no further improvements were required as all the components fit together as required and the overall human fit was comfortable and functional. Overall, the hardware tests (excluding powered elements) were very successful. The iterative process allowed for vital findings to perfect the fit of the brace and improve user experience to meet the project objectives.

9.5.1.2 Electronics with Power Delivery

The objective of these tests was to ensure the PCB could be powered through both the USB Type C connector and through the battery. Powering through the USB was to allow updated code to be flashed to the MCU during software integration, while powering through the batteries was vital for achieving final functionality. As with the physical housings tests detailed above, these were performed in an iterative process – with improvements being made after each, considering any overlapping improvements from other components.

It was during these tests that the battery choices were modified. In the first iteration, the PCB used two battery stack PCBs (see Section 4.4.3) which each contained a single CR2032 coin cell battery capable of delivering 3.0 - 3.3 V. Powering was verified by using a digital multi-meter (DMM) to check correct voltage levels. The findings from the first iteration are shown in Table 18.

Table 18. Hardware Integration Testing: Electronics with Power Delivery - 1st Iteration

Iteration 1			
Test #	Test Performed	Positive	Improvements
10	Safe powering from USB-C connection	No shorts MCU does not heat up	MEMs no power due to circuit schematic mistake – fix using wire bridge, amend on PCB layout
11	Above (#10) correctly powers BLE	Device appears on a Bluetooth detector	-
12	Safe powering from 2x parallel CR2032 batteries	Battery PCBs stack	Header pin position can be adjusted for better stack No power - batteries supply 2.8V, < 3.3V minimum needed
13	Above (#12) correctly powers BLE	-	No power – batteries supply 2.8V, < 3.3V minimum needed

The main improvements after the first iteration were to amend the battery power supply. It was suspected that some voltage was lost on the drop between the two PCB battery stacks; for the second iteration, one of the battery PCBs was discarded, with both enclosures instead soldered onto the same PCB – one on the top side, one on the bottom. This in turn affected enclosure design. The circuit schematic mistake was also fixed. The findings from the second iteration are shown in Table 19 (following page).

From the second iteration, the coin cell batteries were replaced with a single 3.7V rechargeable Lithium Polymer (LiPo) battery. The header pins were moved from the 3.3V line to the USB voltage input line placed before the LDO. To save space due to the battery size increasing, the battery PCBs were discarded, and the enclosure was redesigned to safely hold the new battery. An external power switch was added to the exterior of the enclosure, which was connected to header pins to ensure the battery was fully detachable from the PCB. The findings from the third and final iteration are shown in

Table 20 (following page).

Table 19: Hardware Integration Testing: Electronics with Power Delivery - 2nd Iteration

Iteration 2

Test #	Test Performed	Positive	Improvements
14	Safe powering from USB-C connection	All components receive required power	-
15	Above (#10) correctly powers BLE	Device appears on a Bluetooth detector	-
16	Safe powering from 2x parallel CR2032 batteries	Single PCB fits both batteries	No power - batteries supply 2.8V, < 3.3V minimum needed
17	Above (#12) correctly powers BLE	-	No power – batteries supply 2.8V, < 3.3V minimum needed

Table 20: Hardware Integration Tests: Electronics with Power Delivery - 3rd Iteration

Iteration 3			
Test #	Test Performed	Positive	Improvements
18	Safe powering from USB-C connection	All components receive required power	-
19	Above (#14) correctly powers BLE	Device appears on a Bluetooth detector	-
20	Safe powering from 2x parallel CR2032 batteries	LDO steps down the 3.7V supply to 3.3V All components receive required power	-
21	Above (#16) correctly powers BLE	Device appears on a Bluetooth detector	-

After this iteration, no further improvements to power delivery were required as all components received the correct voltage from both battery and USB, the MCU could be flashed with new code, and the functionality could be verified using a Bluetooth detector.

Overall, hardware tests with and without power were very successful. The iterative process allowed for changes to be made in tandem, with the enclosure being adapted along with the battery supply.

9.5.2 Software Integration

Software integration was centred around the mobile application and ensuring that both the Bluetooth and machine learning code were compatible with it, keeping the application as the overall ‘controller’. Due to the range of programming languages and platforms deployed during the project (Java and Android Studio for the application, C and Espressif-IDE for the MCU, Python and TensorFlow for the ML Model) it was easier to complete integration in small steps during development, rather than use the type of modular testing useful for hardware integration. This allowed for more time for detailed debugging across all three software areas.

By the final stages of development, the following software integrations had been verified as successful:

- Application with MCU – BLE implemented in both, with app able to connect to MCU
- Application with ML Model – Model exported to compatible format, with app able to run in background thread

For more detail on software integration testing, see Section 8.1: ML Model Integration and Section 8.2: Application with Hardware.

9.5.3 Hardware and Software Integration

A similar methodological approach to that applied in hardware integration was followed for integrating the hardware and software elements of the project together. The focus of this testing was to verify, step-by-step, that the full end-to-end system was functional, as illustrated in Figure 58.

MEMS recorded sound → MCU filtering and storing → BLE sending/receiving → Application storing → ML processing and classification.

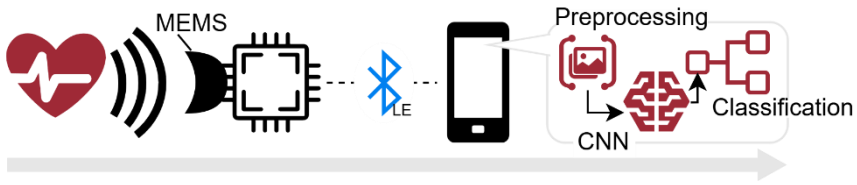


Figure 58. System flow diagram

Tests were performed over 4 main days: Thursday 10th, Friday 11th, Sunday 13th, and Wednesday 16th of April 2025. To keep up with any changes to individual components and ensure no effect on other parts of the system, most tests were repeated on multiple days.

9.5.3.1 PCB with Application – Excl. MCU Filtering

Before this point, the application had been tested using the development board MCU rather than the final PCB – due to earlier issues with power supply. Therefore, the focus of these tests was to ensure that the PCB could successfully connect/disconnect from the app via the MCU, the app could control the operating modes, and some signal could be sent from the MCU to the app. To remove external factors, at this stage no filtering was performed on the MCU, the PCB was not put in the enclosure, and no analysis was performed on the signal once received by the app. The findings are shown in Table 21 (following page).

As the application and BLE interface had been tested through software development using the development MCU board, there were no issues from the software side. Following earlier battery corrections, the PCB also successfully connected to and transmitted information to the app during every day of integration testing.

Next, sample audio files were played for the MEMs to record and signal analysis was performed on the files received by the application. The findings are shown in Table 22 (following page).

Table 21. Hardware and Software Integration: PCB with Application - Signal Transmission

PCB, Mobile Application – Signal Transmission				
Test #	Test Performed	Positive	Improvements	Date
22	MCU connects to app	Connects by button Auto-reconnects	-	10th, 11th, 13th, 16th
23	MCU disconnects from app	Disconnects by button	-	10th, 11th, 13th, 16th
24	App can be used to switch modes on the MCU	App can switch the MCU between 'Standby' and 'Continuous' Verified by checking MCU logs	-	10th, 11th, 13th, 16th
25	Random signal sent by the MCU via BLE	Signal sent	-	10th, 11th, 13th, 16th
26	Random signal received by the MCU via BLE	Appears in 'Recordings' page 'Refresh' button works	-	10th, 11th, 13th, 16th

Table 22. Hardware and Software Integration: PCB with Application - Signal Handling

PCB, Mobile Application – Signal Handling				
Test #	Test Performed	Positive	Improvements	Date
27	MEMs records sample audio	File recorded and stored on MCU	No filtering implemented at this stage	10th
28	MEMs records heart sounds	Placed on mitral point, records a file and stores on MCU	No filtering implemented at this stage	10th, 11th
29	App plays back recording	No audio output, possibly a volume/amplitude issue	Boost values adjusted in MCU code*	10th
30	App plots recording waveform	Plots using Python background tasks	-	10th, 11th, 13th, 16th

*Without MCU filtering, playback was achieved on 10th, 11th, 13th and 16th. With adjusted boost values in MCU, playback was achieved on 11th, 13th and 16th.

9.5.3.2 PCB with Application – Incl. MCU Filtering

The above tests were completed with only the BLE and sampling code present on the MCU. An update with the filtering included was flashed to the MCU, and the tests were repeated. As an easier User Interface (UI) than MCU logs, and verified to work successfully, the application was used to troubleshoot issues with filtering integration. Findings are shown in Table 23.

Table 23. Hardware and Software Integration: PCB with Application - Incl. Filtering

PCB, Mobile Application				
Test #	Test Performed	Positive	Improvements	Date
31	App receives recording from MCU	Appears in 'Recordings'	-	11th
32	App plays back recording	-	App crashes	11th
33	App plots waveform	-	App cannot process .wav file	11th

Without good responses from the application, it was not possible to verify the effects of the filtering. The suspected issue was within the MCU code; previously it had been using integers only but the filtering required floats. This resulted in an improper writing of the .wav file, which in turn crashed the app. The MCU code had to be updated to use floats to continue integration testing.

9.5.3.3 End to End System – Excl. MCU Filtering

While filtering compatibility issues were being resolved (see above), an end-to-end test with the filtering excluded was performed. This involved powering the PCB through batteries, placing all electronic components into the enclosure, placing the enclosure into the brace, and fitting the brace to a group member. A functional, though not yet tuned to filtering, ML model was also uploaded onto the application for full end-to-end testing. The results are shown in Table 24 (following page).

Table 24. End-to-End System Tests - Excl. Filtering

PCB, Enclosure, Brace, Application, ML Model				
Test #	Test Performed	Positive	Improvements	Date
34	All electronics fit in enclosure	Battery fits well with PCB and external switch	-	11th
35	Enclosure fits in brace, positioned over mitral point	Size-adjustable	-	11th
36	PCB powers from battery using external switch	Successful	-	11th
37	App connects and controls recording mode	Successful	-	10th, 11th
38	MEMs records heart sound	Verified with playback, plotting and model recognition	-	10th, 11th
39	App plays back and plots heart sound	Play back includes recognisable heart sound	Plot could be adjusted for quiet volume	10th, 11th
40	Model classifies as expected	Returned ‘healthy’ classification	-	10th, 11th

Below is the screenshot from the app taken during this end-to-end test, showing the heart sound plotted as a basic, unfiltered waveform.

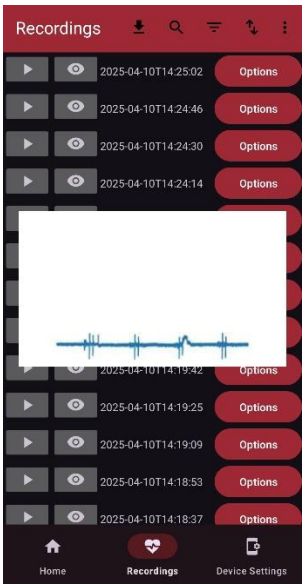


Figure 59. Successful heart sound waveform application screenshot

At this point, the project objectives had been met and all components, excluding MCU filtering, had been integrated successfully.

9.5.3.4 End to End System – Incl. MCU Filtering

Following debugging, the PCB was flashed with corrected MCU code that included filtering, and the end-to-end tests were repeated. Findings are shown in Table 25.

Table 25. End-to-End System Tests - Incl. Filtering

PCB, Filtering, Enclosure, Brace, Application, ML Model				
Test #	Test Performed	Positive	Improvements	Date
41	App connects and controls recording mode	Successful	-	13th
42	MEMs records heart sound	Records a sound	Sounds almost like a heart sound, but unusual	13th
43	App plays back and plots heart sound	-	Plot too flat – missing expected peaks	13th
44	Model classifies as expected	-	Model doesn't recognise as a heart sound, biased to 'abnormal' or 'unknown'	13th

Recordings were taken using the acoustic phantom, an audio speaker, and by placing the PCB on the chest. All recordings sounded very similar – heart 'thumps' seemed present, but they were not as regular as expected, and there was a lot of background noise. To determine if it was a filtering issue or a PCB issue, all tests were repeated using the development board MCU with no filtering. The results from the development board were much closer to expected.

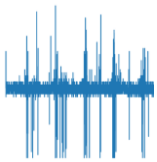
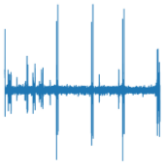
With further tests, it was found that the 'heart sound' was close to the exact output from the MEMs when not directed at any sound at all – i.e. in a quiet environment. The unusual heart sounds were therefore more likely to be noise. Given that the filtering seemed to be working when applied to ML dataset files in MATLAB, it was decided to replace the MEMs microphone to make sure the fragile membrane had not snapped during testing.

9.5.3.5 MCU Filtering Debugging

By the 16th, all other integration testing had been completed. The last test objective was to verify which of the PCB or filtering was functional, if either. Unfortunately, it was not a direct comparison because some components – primarily those around the MEMs - had been removed from the PCB.

The PCB and development board were both flashed with up-to-date filtering code. The PCB had been confirmed to record successfully on the 10th and 11th, but not since. The development board had been confirmed to record successfully at all points. All plots and file sounds were taken from the application, as both boards could connect to it. The same tests using the acoustic phantom, an audio speaker, and the human chest were performed on both. Then the filtering code was removed from both, and the tests were repeated. Findings are shown in Table 26 (following page).

Table 26: Comparison Tests - Filtering and PCB Debugging

PCB, Development Board, MCU Filtering							
Test #	Test Performed*		PCB		Development Board		Date
			Plot:	Sounds like:	Plot:	Sounds like:	
45	MCU Filtering Included	Heart Sound Phantom		Noise		Corrupted noise	16th
46		Heart Sound Speaker		Noise		Corrupted noise	16th
47		Human Chest		Noise		Corrupted noise	16th
48		Background Noise		Noise		Corrupted noise	16th
49	MCU Filtering Excluded	Heart Sound Phantom		Quiet noise		Quiet heart sound	16th
50		Heart Sound Speaker		Quiet noise		Loud heart sound	16th
51		Human Chest		Quiet noise		Quiet heart sound	16th
52		Background Noise		Quiet noise		Quiet noise	16th

*Although not detailed, each test above was performed three-five times each to ensure no rogue noise/interferences

From Table 26, it became clear that there was some error with the MEMs on the PCB. It was also clear that the filtering was unsuccessful, and that the early 'filtered heart sounds' as recorded on the 13th were background noise amplified and adapted to resemble something like a heart sound plot. This was further verified by exporting the recorded filtered files of what was believed to be a heart sound from the human chest (Figure 60) and a background noise recording from the development board (Figure 61) from the application and plotting them with MATLAB.

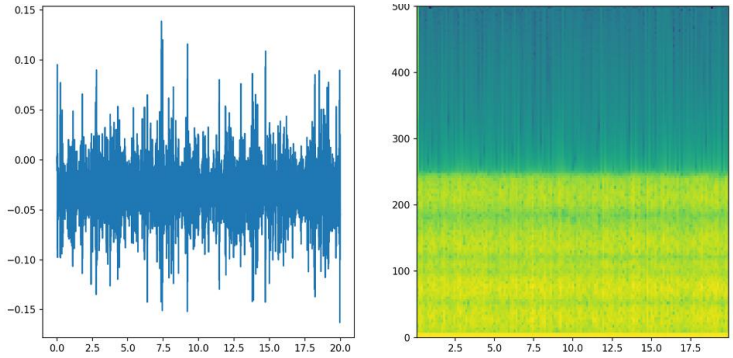


Figure 60: Filtered dev board recording

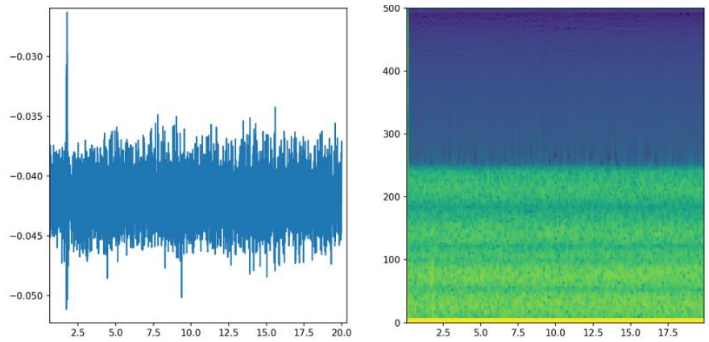


Figure 61: Filtered dev board noise recording

To ensure the project objectives were still met, the MCU filtering was removed from the PCB and the development board. The PCB could still be used to demonstrate all other components and verify them working together – the MEMs being functional was only needed to demonstrate recording. Instead of the MCU, the filtering and denoising would instead come from the static filters in the ML model pre-processing. For more detail on this, see Section 9.7: Additional Tests.

9.5.3.6 MEMs Recordings with ML Model – Phantom Tests

The purpose of these tests was to take the best performing models trained on specific input types (as discussed in Section 8.1.1.6: Grid Validation) and evaluate each model's performance on a test set made up entirely of recordings from the device processed with the paired model pre-processing file. These results would then direct the hyperparameter tuning for the final model to perform better in real-world deployment. The easiest way to perform this test was to export the five models to be tested to. tflite files and pass them to the application back-end along with the five pre-processing files. A simple program could then run each model-processing pair on all of the phantom recordings.

20 audio files were taken from the unseen test set of the model dataset: 10 normal, 10 abnormal. These were played through the acoustic phantom without tissue-mimicking gel, with a thin layer of gel (EcoFlex Gel2 – 10mm), and with a thick layer of gel (EcoFlex Gel2 – 20mm) – totalling 80 audio files across the four categories. The app then ran five pre-processing files on each of the 80 audios to create 400 images fed into the five models. For the purposes of this report, the average accuracy has been taken for each category and model. Results are shown in Table 27 (following page).

Table 27: ML Model Tests - Phantom

Phantom Configuration	ML Model				
	Model 1	Model 2	Model 3	Model 4	Model 5
Control	90%	75%	60%	85%	85%
Phantom – No Gel	47%	65%	47%	70%	75%
Phantom – Thin Gel	50%	75%	75%	70%	50%
Phantom – Thick Gel	50%	70%	50%	50%	50%

It should be noted that a significant issue for the ML Model at this stage was the lack of digital filtering performed by the MCU. While some training data was corrupted with noise, most contained clean, clear signals. By contrast, the unfiltered MEMs recordings were much noisier. This is why in the grid validation – rather than test the impact of different implementations of the MCU digital filter – it instead compares the impact of various bandpass filters. It is also why Model 1, despite having the highest accuracy of 90% on the control set (i.e. not real-world), performed very poorly in the following sets. Model 1 followed the base CNN architecture and had no filtering beyond a simple 25-500Hz bandpass; relying on the MCU to take the brunt of denoising the recordings. Models 2, 3, 4 and 5 – each with more extensive bandpass and other static filters – performed better in real-world deployment, with the thin-gel phantom producing the most consistent results at around 75%.

The synthetic gel chosen for the phantom, EcoFlex Gel2, is used in medical applications to mimic tissue. Including it in the phantom helped to ensure the heart sounds vibrated upwards, better mimicking the chest cavity. As the thin gel performed best in the ML Model phantom tests when compared to the control/dataset only tests, the thin gel was closest to the chest cavity effects. Therefore, the model to be tuned from this point was Model 2.

9.5.3.7 MEMs Recordings with ML Model – Human Tests

The last stage of integration testing concerned a user wearing the device. It was important to test the device in various states of activity, as it was designed to be wearable for long periods of time.

These tests were performed using Model 2 with all filtering taking place in the model pre-processing. For recording, the development board was placed inside a slightly modified enclosure inside the brace. The mode was set to ‘continuous’, with the device kept in place while seven recordings were taken. The first and last recording were discarded to eliminate any accidental noise from switching the device on/off, leaving five classifications for each activity. As tests were performed on a group member, every recording was expected to come back as ‘normal’. Results are shown in Table 28.

Table 28: ML Model Tests - Human

Activity	Classification	
	Normal	Abnormal
Sitting	100%	0%
Standing	80%	20%
Walking	100%	0%
Talking	60%	40%

The results for sitting and standing were about as expected with the model performance on the phantom. Interestingly, the model did not seem to struggle with movement and walking but did significantly when the user talked throughout the recordings. This may be due to the frequency difference between the two activities and would be good to look at in further work.

9.6 TESTING CONCLUSIONS

Overall, while not entirely successful, the integration testing provided valuable insights into the device functionality. The fit of the brace and overall hardware integration were successfully optimised. The final design is comfortable, intuitive, and able to be fitted on and adjusted by a user themselves. While not tested beyond group members, the adjustable brace design should accommodate for users of most body types.

The PCB successfully communicates with the mobile application and can send data to it over BLE. In one set of end-to-end tests, the PCB also successfully recorded a heart sound from a group member using the MEMS microphone. Further investigation into why the final MEMS is not recording/not sending recorded data to the MCU should be performed. Additional schematic tests and simulations could be performed to determine whether it is an issue with the MEMS itself or the schematic design. Insights were gained into digital filtering, although further tests to understand exactly what caused the erroneous outputs should be performed.

The ML Model was successfully tested on recordings gathered from the development board MCU and MEMS, using all three configurations of the acoustic phantom as well as recording from the human chest. The model developed through purely dataset tests and tuning was optimised for real-world deployment by comparing different pre-processing methods (spectrogram types, sampling rates, duration, etc.) and adjusting model architectures (no. of layers, kernel size, learning rate, etc.) to get the highest accuracy rate. Additional filtering was also added into the ML pre-processing to compensate for the lack of MCU filtering and mitigate the effect this had on overall model accuracy.

Finally, some test results were gathered on a group member performing daily activities (sitting, standing, walking, and talking) in a lab setting. This provided additional insights into situations where the device struggles to find the required heart sound, such as the lowest performance when the user talks. This is something that could be addressed in future project iterations. The results of the full integration testing are compared with the high-level test plan (Table 13) in the below table (Table 29, following page).

Table 29: Test Plan Summary of Results

Test Objective	Testing Subject	Results Interpretation
Can it record a heart sound?	Group Members	Comparison of recorded signal to expected signal (from data sets or other heart measurement device)
Can it identify abnormal heart sounds?	Acoustic Phantom	Comparison of heart sound classification to actual heart sound classification
Can it record suitable heart sounds in noisy environment?	Group Members	Qualitative assessment of signal
To what extent can it identify abnormal heart sounds in noisy environments?	Acoustic Phantom	Comparison of heart sound classification to actual heart sound classification
Can it record suitable heart sounds while user is performing common activities?	Group Members	Qualitative assessment of signal, checking for motion artifacts

9.7 ADDITIONAL TESTS

Due to some elements of integration testing being unsuccessful with insufficient time to adequately fix them there are additional tests that were not performed. This section details those tests, along with other important tests that would be required if this product were to eventually be used in real-world applications. It will also detail why these tests are important.

9.7.1 Resolving Errors

The first step of any additional tests would be to return the PCB to a functional state and find and solve the filtering issue. It would be useful to run more tests and simulations with the circuit schematic first, to verify there are no underlying issues with the circuitry. Assuming no changes, the final PCB layout could then be sent to an external manufacturer (e.g. JLC or PCBTrain), for the advantage of copper-plated vias, multiple layers, and for each component to be professionally placed onto the board. External manufacture was in the project plan, but the long lead times combined with integration issues did not allow for it.

Significant research on filtering has been conducted throughout the project (see Section 4.2:Digital Signal Processing (DSP)) and the ATF has been evidenced in MATLAB to clean up noisy signals to a useful degree. Therefore, fixing the filtering would likely be a case of trialling different configurations and parameters, to see what improved the results. Going back to researched theory would support the selection of these parameters. Finally, the most useful change would be to conduct phantom and human-based tests throughout filter development to ensure any changes maintain compatibility.

9.7.2 Repeating Development Board Tests on PCB

Due to integration issues, some of the tests were completed on the development board rather than the PCB – as the development board MEMS was proven to work. Given more time and the PCB returned to a functional state, they should be repeated with the PCB to keep the overall results as accurate as possible to the device. This would also better influence any future design changes.

9.7.3 Investigating Power Consumption

For a continuous monitoring device, measuring power consumption is vital. Using values obtained through calculation, the battery should last for one day of continuous heart sound sampling before needing to be recharged. A useful test would be to verify this by leaving the device on continuous mode for as long as possible to measure maximum battery life; the same could be done for the low and standby modes. The low power mode is designed to use less and extend battery life – it would be useful to have tested for confirmation of this.

9.7.4 Application Usability Testing

Conducting usability testing with the target end users of the system would be very useful from the software side, as end users are often able to make insights and point out features/issues that developers or designers miss. This would help to rectify any interface issues between the user and the application. It would also allow for confirmation that the app design – intended to be simple and easy-to-navigate – met its main design goals.

9.7.5 Different Noise Environments

The goal was to use adaptive-noise filtering in the MCU, in which the filter removes the average noise response present in each sample. This was to give the project device an edge over similar devices on the market/in production – a successful at-home monitoring system would be able to filter out noise produced from daily activities such as walking, talking, or light exercise. It should also be able to filter out background noise in a range of environments, to mirror where the user might go in daily life. Useful tests would be to use the acoustic booth in the TIC to measure a zero-noise floor, then manually adjust the noise level present in the room to obtain a wide range of device responses in different artificial environments. With filtering improvements from that lab testing, a similar process could then be carried out taking the device into real-world settings with varying noise levels.

9.7.6 Different Brace Requirements

For proper application with as wide a range of users as possible, the brace should be adjustable for a wide range of people and body types. To achieve this, it should be tested on as many different body types as possible. In project development, this was limited to only the group members due to time constraints and ethical considerations about involving anyone not in the project for testing. Aside from comfort and general fit, it would be useful to ensure that the device always sat on the user's mitral point, regardless of their body type.

10 COMMERCIAL ANALYSIS

10.1 MARKET COMPARISON

The aim of this project was to design and develop a wearable, at-home, long-term monitoring device for cardiac auscultation. As part of that goal, the device needed to be affordable to the average end user, be able to perform as expected, and fill a useful niche in the current market. The niche this project was aiming to fill was that of a device that was as non-invasive and accessible as possible, that was easy to use at home and without expert supervision. Of the similar products and devices that exist already, the five most relevant were the conventional stethoscope, Implantable Loop Recorder (ILR), portable Holter monitor, StethoMe, and Tytocare. Below in Table 30 is shown a market comparison of these products to the device developed through this project (referred to as WCAAMS).

Table 30. Market Comparison

Product	**Description**	**End Users**	**Purpose**	**Accessibility**	**Limitations**
Conventional Stethoscope	Device to listen to heart sounds in real-time	Clinicians	Clinician care and diagnosis	Good – analogue device, simple to use	No digital aspects means no data storage, processing or automation. Requires clinician expertise to use properly
Implantable Loop Recorder (ILR)	Device inserted under skin to record electrical signals from heart (ECGs)	Clinicians, high-risk patients	Long-term arrythmia detection	Poor – implantable, requires surgery	Invasive, requires surgery and clinician expertise. Often expensive
Portable Holter Monitor	Machine with electrodes, or worn as a patch, to measure electrical signals (ECGs)	Patients	Short-term arrythmia detection	Poor – bulky	Can be bulky and/or expensive
StethoMe	AI device for the at-home monitoring of lung sounds	Patients, families	At-home monitoring	Good – wireless device that uses AI	-
TytoCare	Clinician guided remote auscultation	Patients, families, telehealth providers	Telehealth	Good – wireless device with telehealth	Requires clinician expertise. Not suitable for long-term conditions, not wearable. Dependant on an internet connection
WCAAMS	AI device for the at-home monitoring of heart sounds	Patients	Long-term, at-home monitoring	Good – wireless device that uses AI	Struggles with some daily activities

Looking at the market, there is a niche for long-term, at-home monitoring systems, at the moment populated by portable devices such as StethoMe and telehealth systems like TytoCare. The key benefit is that these systems do not require clinician expertise to be functional – systems like StethoMe instead make use of AI to aid with result classifications. The WCAAMS project device was inspired by the StethoMe and sticks close to the key features that made the StethoMe successful, while expanding on it to highlight cardiac auscultation and the function of a wearable device.

10.2 COST OF ONE DEVICE

The cost for manufacture of one device was estimated using component BoMs, average hardware manufacturing prices, and average cost of physical materials (3D printing polymer, brace elastic, etc.) Each main component and how it contributed to the cost of one device is shown in Table 31.

Table 31. Calculating Cost of One Device

Item	Area of Device	Component	Quantity	Sourcing Costs (est.)	Manufacturing Costs (est.)
1	Enclosure	3D printed backplate	1	1.50	2.00
2	Enclosure	3D printed cover	1	2.00	2.00
3	Enclosure	Screws	4	0.20	-
4	Brace	Elastic bands	1	5.40	20.00
5	Brace	Fabric	1	1.00	-
6	Brace	Adjustable clips	2	0.50	-
7	Electronics	ESP32-C6H8	1	4.00	-
8	Electronics	ICs	3	3.70	-
9	Electronics	Res/Caps	20	4.60	-
10	Electronics	2-sided PCB	1	10.00	20.00
11	Electronics	3.7V Battery	1	3.50	-
			Sum	36.40	44.00
				Total	**£ 80.40**

11 CONCLUSIONS

12 FURTHER WORK

This section details any further work that has been identified. This could be work that was intended to be completed but ended up being out of scope, add-ons to improve the product, or just generally expand on what was detailed in this report. Notably this excludes additional testing as this is covered in Section 9.7.

12.1IMPROVING SECURITY & USER EXPERIENCE

To make the system easier for an end user and to improve their confidence in using the device the application should provide visual instructions on how to fit the brace and position the device. While there is a User Guide (supplied in Additional Materials) having the instructions in the app would be easier for many users. The application could also provide dynamic feedback/calibration to help the user ensure the device is in the optimal position, possibly by listening to heart sounds and providing feedback on how loud/clear they are.

The notification screen, identified during the requirements phase, should also be implemented. This would update users when an arrythmia was detected and provide a link the recording in which it occurred. Users could then more easily see and identify patterns in their health data.

To improve security, data encryption should be used. This would be in two stages, with the most important being while the data is being transmitted over Bluetooth. Then also while it is stored on the phone itself. This would prevent anyone from accessing patient's private health data. The application could also provide a feature that requires a password/pin to open the app. These features would be vital if the application were to store patient information like diagnoses to monitor.

Additionally, power analysis and improvement should be performed. This would allow the device to function for longer without being charged and ideally eventually allow the system to switch back to the originally intended watch/cell batteries. These would be easier and safer for users to handle.

12.2 EXPANDING USE CASES

The system could also be expanded to include additional use cases. Firstly, Telehealth – automatically sending recorded and relevant results to healthcare providers. Another use would be to provide current episode detection. However, this would require careful testing to the ML Model to ensure it has an extremely low false positive rate. It would also require specific arrythmias to be classified as some can be harmless.

The device is currently capable of long-term monitoring as users can wear it as often as they like. However, the application could provide additional support such as allowing users to input health information and diagnoses to track. The app could then provide feedback relevant to the condition they are tracking.

The MEMS microphone at the auscultation point used should be capable of picking up lung sounds. Further research could be done into how to distinguish heart and lung sounds from each other with filtering and ML Model training. This would then allow a whole other range of diseases to be detected and monitored.

13 REFERENCES

[1] World Heart Federation (WHF), “World Heart Report 2023: Confronting the World's Number One Killer,” World Heart Federation, Geneva, Switzerland, 2023.

[2] British Heart Foundation (BHF), “Heart Statistics: The UK in numbers (BHF Statistics Factsheet - UK),” September 2024. [Online]. Available: https://www.bhf.org.uk/what-we-do/our-research/heart-statistics. [Accessed 17 10 2024].

[3] Oxford Population Health, “Urgent action is required to stem the increasing costs of major conditions,” 25 July 2024. [Online]. Available: https://www.ndph.ox.ac.uk/news/urgent-action-is-required-to-stem-the-increasing-costs-of-major-conditions. [Accessed 17 10 2024].

[4] “Private 'at home' ECG monitoring,” Yourheartscan, [Online]. Available: https://www.yourheartscan.co.uk/ambulatory-ecg-monitoring. [Accessed 2024 April 21].

[5] GetWellue, Wellue, [Online]. Available: https://getwellue.com/products/heart-health-monitor. [Accessed 2024 April 21].

[6] “KardiaMobile,” ALIVECOR, [Online]. Available: https://store.alivecor.co.uk/products/kardiamobile. [Accessed 21 April 2024].

[7] iRhythm, [Online]. Available: https://www.irhythmtech.com/gb/en/solutions-services/the-proven-ambulatory-cardiac-monitoring-service. [Accessed 2025 April 21].

[8] NICE , “Case study: Zio XT - From NICE digital health pilot to AI Award winner,” NICE , [Online]. Available: https://case-studies.nice.org.uk/zio-xt/index.html. [Accessed 2025 April 21].

[9] T. Grzywalski, M. Piecuch and M. Szajek, “Practical implementation of artificial intelligence algorithms in pulmonary auscultation examination,” vol. 883–890, 29 March 2019.

[10] O. Haskel, E. Itelman, E. Zilber, G. Barkai and G. Segal, “Remote Auscultation of Heart and Lungs as an Acceptable Alternative to Legacy Measures in Quarantined COVID-19 Patients—Prospective Evaluation of 250 Examinations,” vol. 3165, 20 April 2022.

[11] ,. J. C. C. H. G. W. C. Haoran Ren, “A Novel Cardiac Auscultation Monitoring System Based on Wireless Sensing for Healthcare,” IEEE Journal of Translation Engineering in Health and Medicine , 2018.

[12] R. S. Downen, B. Li, Q. Dong and Z. Li, “Cloud-Connected Patch-Worn Auscultation Device for Chest Sound Monitoring,” 5 07 2024.

[13] W. Y. Shi, J. Mays and J.-C. Chiao, “Wireless stethoscope for recording heart and lung sound,” *IEEE Topical Conference on Biomedical Wireless Technologies, Networks, and Sensing Systems (BioWireleSS)*, 2016.

[14] S. Swarup and A. N. Makaryus, “Digital stethoscope: technology update,” *Medical Devices: Evidence and Research*, vol. 11, 2018.

[15] S. H. Lee, Y.-S. Kim, M.-K. Yeo, M. Mahmood, N. Zavanelli and C. Chung, “Fully portable continuous real-time auscultation with a soft wearable stethoscope designed for automated disease diagnosis,” *Science Advances,* vol. 8, no. 21, 2022.

[16] B. L. Q. D. a. Z. L. R. Scott Downen, “Cloud-Connected Patch-Worn Auscultation Device for Chest Sound Monitoring,” 2024 .

[17] I. McLane, D. Emmanouilidou, J. E. West and M. Elhilali, “Design and Comparative Performance of a Robust Lung Auscultation System for Noisy Clinical Settings,” *Journal of Biomedical and Health Informatics,* vol. 25, no. 7, pp. 2583 - 2594, 2021.

[18] Y. Cotur, “ Stretchable Composite Acoustic Transducer for Wearable Monitoring of Vital Signs,” *Advanced Functional Materials,* vol. 30, no. 15, 2020.

[19] J. Jusak and I. Puspasari, “Wireless tele-auscultation for phonocardiograph signal recording through Zigbee networks,” *IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob),* 2015.

[20] Y. Lv, M. P. I. Dias, L. Ruan, E. Wong, Y. Feng and N. Jiang, “Request-Based Polling Access: Investigation of Novel Wireless LAN MAC Scheme for Low-Latency E-Health Applications,” *IEEE Communications Letters,* vol. 23, no. 5, pp. 896 - 899, 2019.

[21] A. Meiappane, S. S. Murugan, A. Arun and A. Ramachandran, “Latency of Web Service in Health Care System Using GSM Networks,” *Second International Conference on Machine Learning and Computing,* 2010.

[22] R. Istepanian, E. Jovanov and Y. Zhang, “Guest Editorial Introduction to the Special Section on M-Health: Beyond Seamless Mobility and Global Wireless Health-Care Connectivity,” *IEEE Transactions on Information Technology in Biomedicine,* vol. 8, no. 4, pp. 405 - 414, 2004.

[23] B. S. Emmanuel, “A review of signal processing techniques for heart sound analysis in clinical diagnosis,” *Journal of Medical Engineering & Technology,* vol. 36, no. 6, pp. 303-307, 2012.

[24] N. S. Haider, “Feature Extraction and Classification Methods for Lung Sounds,” *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* vol. 10, no. 1, pp. 128-137, 2020.

[25] S. B. Patel, T. F. Callahan, M. G. Callahan, J. T. Jones, G. P. Graber, K. S. Foster, K. Glifort and G. R. Wodicka, “An adaptive noise reduction stethoscope for auscultation in high noise environments,” *J. Acoust. Soc. Am,* vol. 103, p. 2483–2491, 1998.

[26] O. Haskel, E. Itelman, E. Zilber, G. Barkai and G. Segal, “Remote Auscultation of Heart and Lungs as an Acceptable Alternative to Legacy Measures in Quarantined COVID-19 Patients—Prospective Evaluation of 250 Examinations,” *Sensors 2022,* vol. 22, no. 9, pp. 1424-8220, 2022.

[27] I. Müller, J. Henze, A. Burmann, R. Salvi, M. Friebe and R. Baum, “Home Monitoring of the Carotid Arteries Using a Mobile Auscultation Device with App”.

[28] I. Muller, “Home Monitoring of the Carotid Arteries,” *Current Directions in Biomedical Engineering,* vol. 8, no. 2, pp. 544-547, 2022 .

[29] V. Jiménez-Mixco, M. F. Cabrera-Umpiérrez, M. T. Arredondo, A. M. Panou, M. Struck and S. Bonfiglio, “Feasibility of a wireless health monitoring system for prevention and health assessment

of elderly people," *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC),* 2013.

[30] P. Phatiwuttipat, W. Kongprawechon, K. Tungpimolrut and S. Yuenyong, "Cardiac auscultation analysis system with Neural Network and SVM technique," *The 8th Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology (ECTI) Association of Thailand - Conference 2011,* 2011.

[31] T. Grzywalski, M. Piecuch, M. Szajek, A. Bręborowicz, H. Hafke-Dys, J. Kociński, A. Pastusiak and R. Belluzzo, "Practical implementation of artificial intelligence algorithms in pulmonary auscultation examination," *European Journal of Pediatrics,* vol. 178, p. 883–890, 2019.

[32] Z. Sankari and H. Adeli, "HeartSaver: A mobile cardiac monitoring system for auto-detection of atrial fibrillation, myocardial infarction, and atrio-ventricular block," *Computers in Biology and Medicine,* vol. 41, no. 4, pp. 211-220, 2011.

[33] C. Doukas, I. Maglogiannis, V. Koufi, F. Malamateniou and G. Vassilacopoulos, "Enabling data protection through PKI encryption in IoT m-Health devices," *2012 IEEE 12th International Conference on Bioinformatics & Bioengineering (BIBE),* 2012.

[34] StethoMe, "StethoMe - lungs under control," 2018. [Online]. Available: https://www.stethome.com/en-gb.

[35] J. Ko, C. Lu, M. B. Srivastava, J. A. Stankovic, A. Terzis and M. Welsh, "Wireless Sensor Networks for Healthcare," *Proceedings of the IEEE,* vol. 98, no. 11, pp. 1947 - 1960, 2010.

[36] R. Istepanian, E. Jovanov and Y. Zhang, "Guest Editorial Introduction to the Special Section on M-Health: Beyond Seamless Mobility and Global Wireless Health-Care Connectivity," vol. 8, no. 4, pp. 405 - 414.

[37] J. Jusak and I. Puspasari, "Wireless tele-auscultation for phonocardiograph signal recording through Zigbee networks," *2015 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob),* 11 01 2016.

[38] W. Shi, J. Mays and J.-C. Chiao, "Wireless stethoscope for recording heart and lung sound," 04 04 2016.

[39] S. Ray, "Essentials of Machine Learning Algorithms (with Python and R Codes)," Analytics Vidhya, 22 11 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/.

[40] S. T. A. N. Y. Zeinali, "'Heart sound classification using signal processing and machine learning algorithms'," *Machine Learning with Applications,* no. https://doi.org/10.1016/j.mlwa.2021.100206.

[41] W. Y. Shi, J. Mays and J.-C. Chiao, "Wireless stethoscope for recording heart and lung sound," *IEEE Topical Conference on Biomedical Wireless Technologies, Networks, and Sensing Systems (BioWireleSS),* 2016.

[42] G. D. Caterina, *EE580 course notes,* Glasgow, 2025.

[43] A. Kazemnejad, P. Gordany and R. Sameni, “EPHNOGRAM: A Simultaneous Electrocardiogram and Phonocardiogram Database,” 2021.

[44] espressif, “GitHub: BLE Heart Rate Measurement Example,” 25 June 2019. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/nimble/blehr. [Accessed 24 January 2025].

[45] “esp-nimble-cpp,” [Online]. Available: https://h2zero.github.io/esp-nimble-cpp/index.html. [Accessed 2025 4 20].

[46] D. Haughty, “MoSCoW Method,” Project Smart, 15 10 2021. [Online]. Available: https://www.projectsmart.co.uk/tools/moscow-method.php. [Accessed 23 11 2024].

[47] A. A. Alanazi, S. R. Atcherson, C. A. Franklin and M. F. Bryan, “Frequency Responses of Conventional and Amplified Stethoscopes for Measuring Heart Sounds,” *Saudi Journal of Medicine and Medical Sciences.*

[48] W. Chen and e. al, “'Classifying Heart-Sound Signals Based on CNN Trained on MelSpectrum and LogMel Features',” no. https://pmc.ncbi.nlm.nih.gov/articles/PMC10294824/-.

[49] J. Parak, “Comparison of Heart Rate Monitoring Accuracy between Chest Strap and Vest during Physical Training and Implications on Training Decisions,” *sensors,* vol. 21, no. 24, 2021.

[50] D. Spierer, “Validation of photoplethysmography as a method to detect heart rate during rest and exercise,” *Journal of Medical Engineering & Technology* , vol. 39, no. 5, pp. 264-271, 2015.

[51] H. Austerlitz, “Nyquist Theorem - An Overview,” *ScienceDirect Topics,* no. https://www.sciencedirect.com/topics/engineering/nyquist-theorem.

[52] Google AI for Developers, “LiteRT Overview,” Google, [Online]. Available: https://ai.google.dev/edge/litert.

[53] S. D. K. Kota, “Running ML Models in Android using Tensorflow Lite,” Medium, 27 07 2020. [Online]. Available: https://medium.com/analytics-vidhya/running-ml-models-in-android-using-tensorflow-lite-e549209287f0. [Accessed 15 03 2025].

[54] A. Raj, “ML in Android -3 : Deploy Tflite on Android using Java,” Medium, 24 01 2022. [Online]. Available: https://medium.com/the-stem/ml-in-android-3-deploy-tflite-on-android-using-java-6a18431644b6. [Accessed 15 03 2025].

[55] Google AI for Developers, “Model,” 10 05 2024. [Online]. Available: https://ai.google.dev/edge/api/tflite/java/org/tensorflow/lite/support/model/Model#run(java.lang.Object[],%20java.util.Map%3Cjava.lang.Integer,%20java.lang.Object%3E). [Accessed 16 03 2025].

[56] Chaquopy, “Gradle plugin,” [Online]. Available: https://chaquo.com/chaquopy/doc/current/android.html. [Accessed 10 03 2025].

[57] Chaquopy Github, “Update numba (for librosa): Issue #834,” 27 03 2023. [Online]. Available: https://github.com/chaquo/chaquopy/issues/834. [Accessed 13 03 2025].

[58] SMOOTH-ON, “Ecoflex™ GEL 2,” SMOOTH-ON, [Online]. Available: https://www.smooth-on.com/products/ecoflex-gel-2/. [Accessed 2025 April 22].

[59] Infineon, “IM73A135V01 datasheet”.

[60] S. Kang, A. Clayton and S. Wojcik, “Ventricular Tachycardia (VT),” Cedars Sinai, [Online]. Available: https://www.cedars-sinai.org/health-library/diseases-and-conditions/v/ventricular-tachycardia-1.html#:~:text=Ventricular%20tachycardia%20(VT)%20is%20a,it%20can%20become%20life%2Dthreatening. [Accessed 17 10 2024].

[61] M. A. Shiwani and e. al, “Continuous monitoring of health and mobility indicators in patients with cardiovascular disease: a review of recent technologies,” White Rose, 2023.

[62] British Heart Foundation (BHF), “Implantable Loop Recorders,” [Online]. Available: https://www.bhf.org.uk/informationsupport/tests/implantable-loop-recorders. [Accessed 17 10 2024].

[63] A. Rohr, “Everything Med Students Need to Know About S1, S2, S3, & S4 Heart Sounds,” MedStudy, 01 02 2024. [Online]. Available: https://explore.medstudy.com/blog/s1-s2-s3-s4-heart-sounds.

[64] Q. Zhao, S. Geng, B. Wang, Y. Sun and a. S. Hong, “Deep Learning in Heart Sound Analysis: From Techniques to Clinical Applications,” 09 10 2024. [Online]. Available: https://spj.science.org/doi/10.34133/hds.0182.

[65] S. Weiss, *EE578 Course Notes*, 2024.

[66] PhysioNet, “About PhysioNet,” 2018. [Online]. Available: https://archive.physionet.org/about.shtml. [Accessed 07 12 2024].

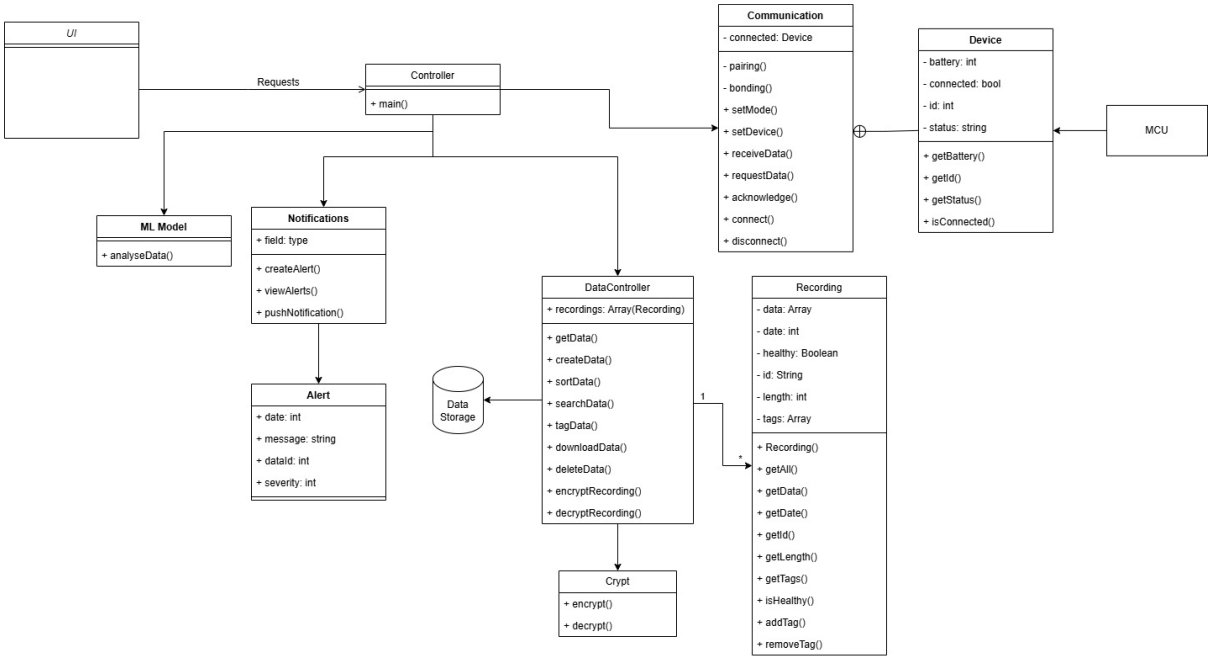
[67] “ThePiHut,” ThePiHut, [Online]. Available: https://thepihut.com/products/esp32-c6-mini-development-board. [Accessed 02 12 2024].

[68] Infineon Technologies, “IM73A135V01: IP57 dust and water resistant analog XENSIV MEMs microphone,” [Online]. Available: https://www.digikey.co.uk/en/products/detail/infineon-technologies/IM73A135V01XTSA1/15776498?srsltid=AfmBOoo1dnqUHGn-_6gUa3_H2FL0WPYD2E5TTEsU3cttvXkcChEpowZM.

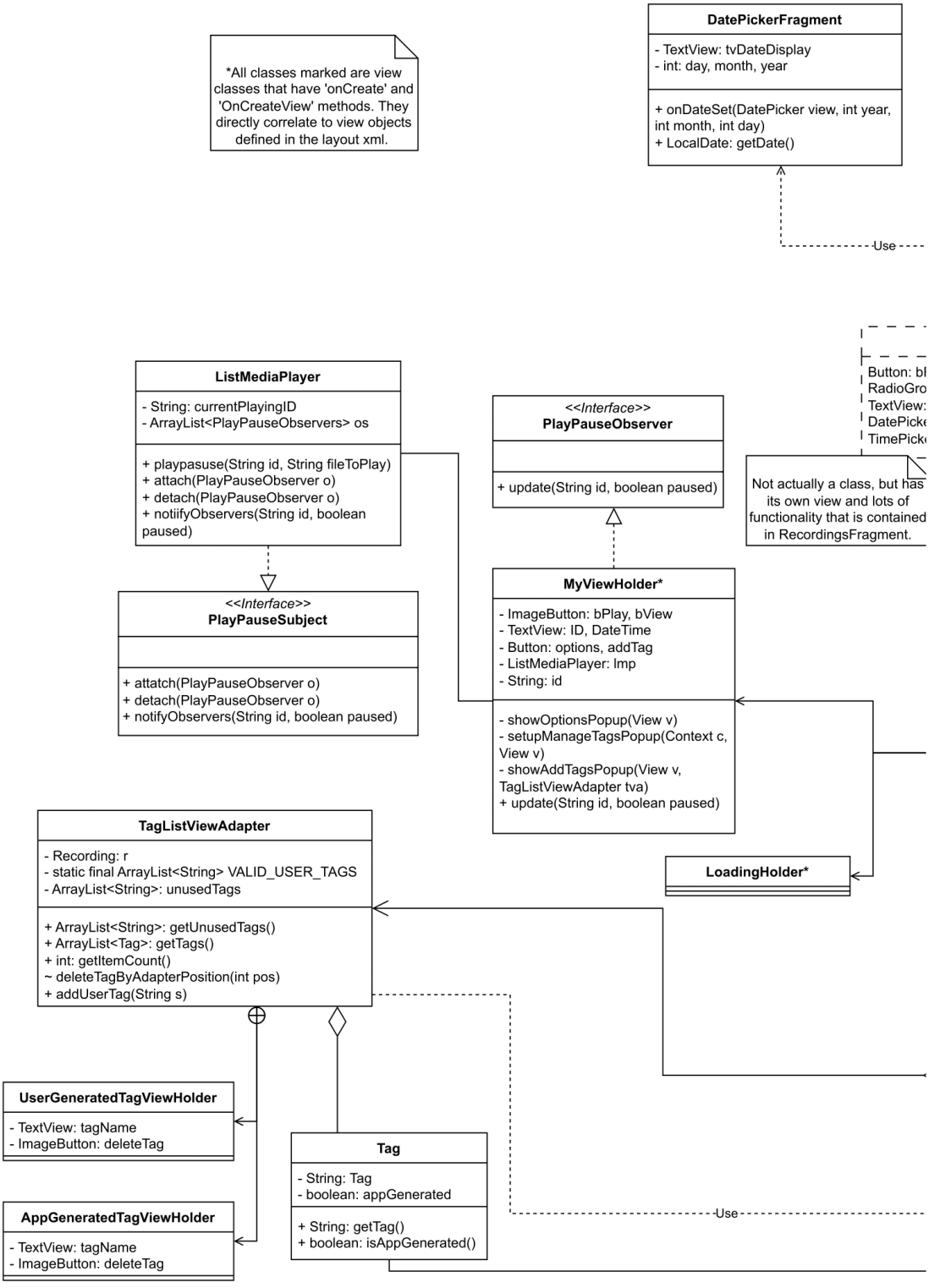
14 APPENDICES

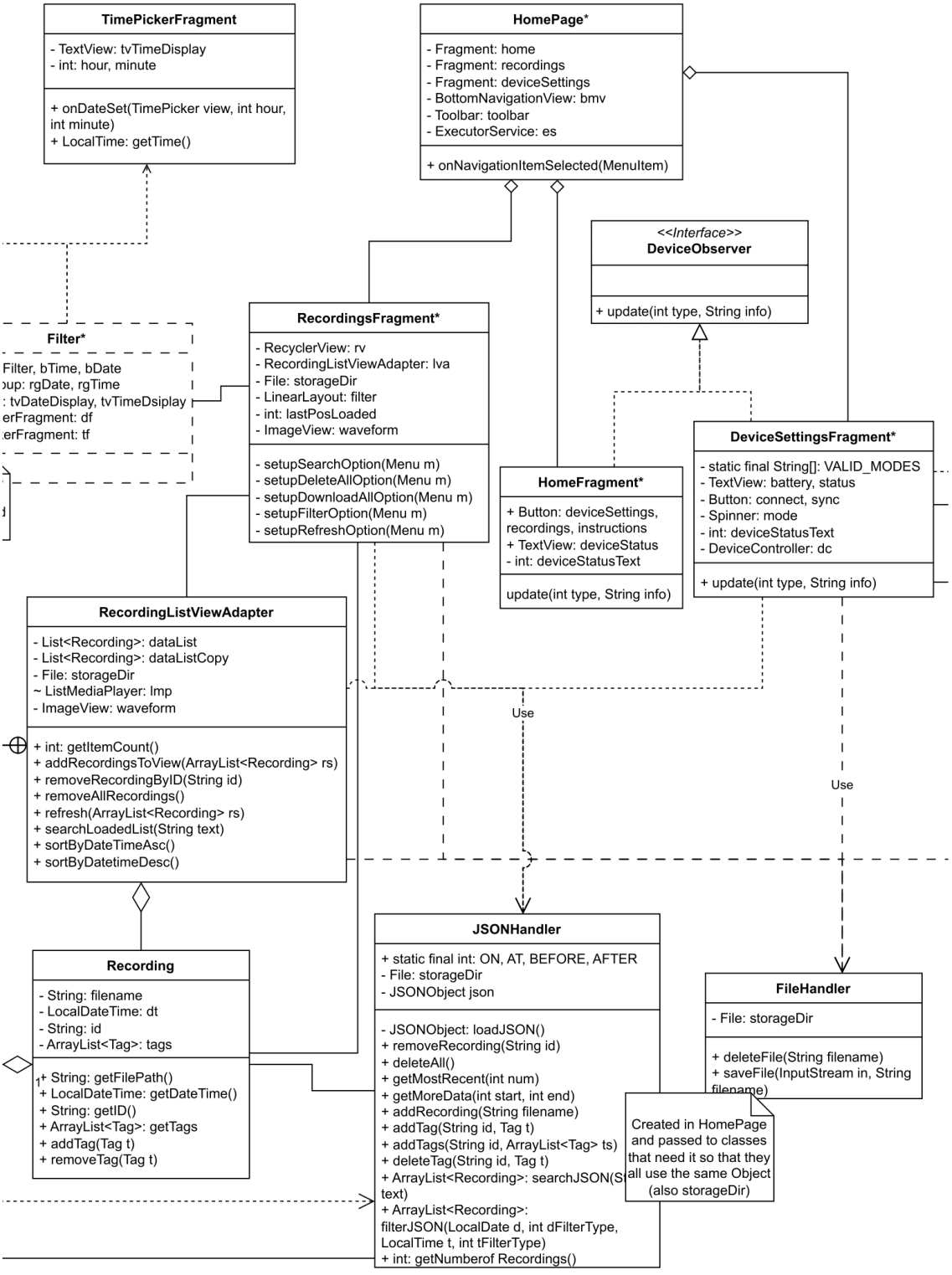
A. APPLICATION CLASS DIAGRAMS

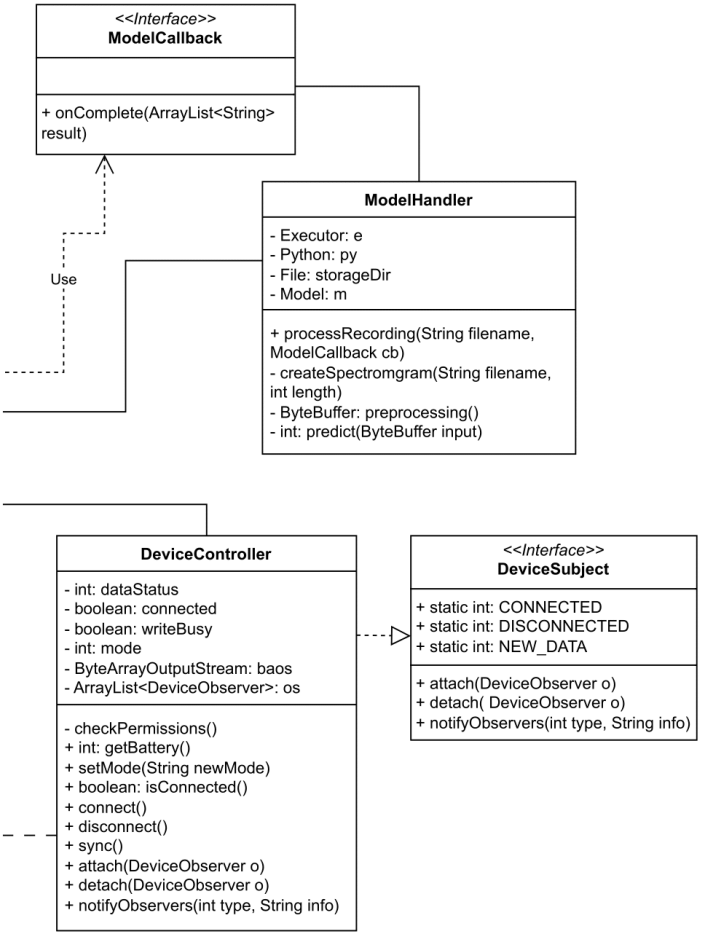
Initial Design:



Final Design, full size:

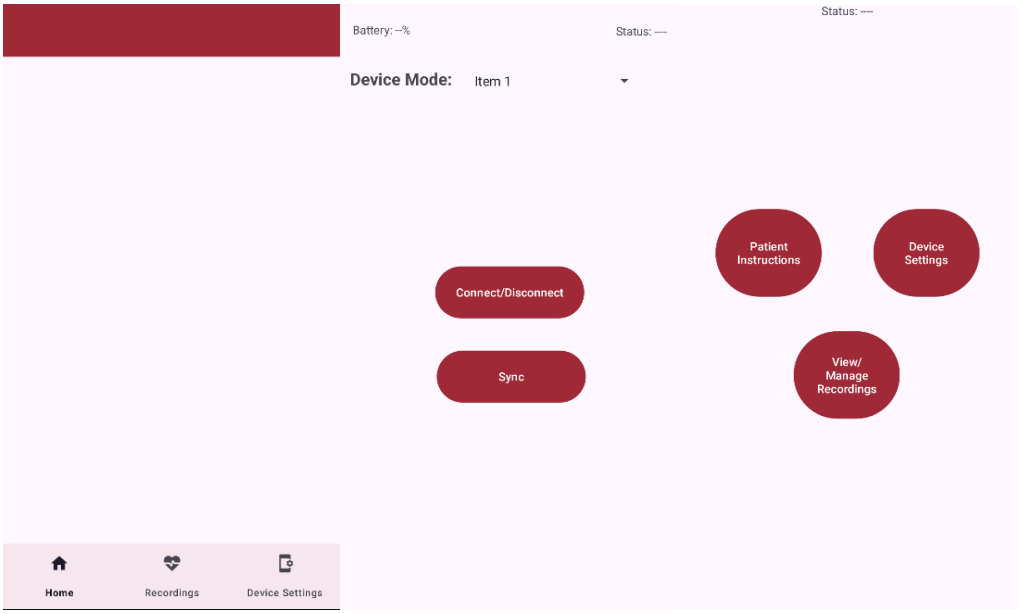

*All classes marked are view classes that have 'onCreate' and 'OnCreateView' methods. They directly correlate to view objects defined in the layout xml.
DatePickerFragment
- TextView: tvDateDisplay
- int: day, month, year
+ onDateSet(DatePicker view, int year, int month, int day)
+ LocalDate: getDate()
Use
ListMediaPlayer
- String: currentPlayingID
- ArrayList<PlayPauseObservers> os
+ playpasuse(String id, String fileToPlay)
+ attach(PlayPauseObserver o)
+ detach(PlayPauseObserver o)
+ notiifyObservers(String id, boolean paused)
<<Interface>>
PlayPauseObserver
+ update(String id, boolean paused)
Button: bI
RadioGro
TextView:
DatePicke
TimePick
Not actually a class, but has its own view and lots of functionality that is contained in RecordingsFragment.
<<Interface>>
PlayPauseSubject
+ attatch(PlayPauseObserver o)
+ detach(PlayPauseObserver o)
+ notifyObservers(String id, boolean paused)
MyViewHolder*
- ImageButton: bPlay, bView
- TextView: ID, DateTime
- Button: options, addTag
- ListMediaPlayer: lmp
- String: id
- showOptionsPopup(View v)
- setupManageTagsPopup(Context c, View v)
- showAddTagsPopup(View v, TagListViewAdapter tva)
+ update(String id, boolean paused)
TagListViewAdapter
- Recording: r
- static final ArrayList<String> VALID_USER_TAGS
- ArrayList<String>: unusedTags
+ ArrayList<String>: getUnusedTags()
+ ArrayList<Tag>: getTags()
+ int: getItemCount()
~ deleteTagByAdapterPosition(int pos)
+ addUserTag(String s)
LoadingHolder*
UserGeneratedTagViewHolder
- TextView: tagName
- ImageButton: deleteTag
AppGeneratedTagViewHolder
- TextView: tagName
- ImageButton: deleteTag
Tag
- String: Tag
- boolean: appGenerated
+ String: getTag()
+ boolean: isAppGenerated()
Use


TimePickerFragment
- TextView: tvTimeDisplay
- int: hour, minute
+ onDateSet(TimePicker view, int hour, int minute)
+ LocalTime: getTime()
HomePage*
- Fragment: home
- Fragment: recordings
- Fragment: deviceSettings
- BottomNavigationView: bmv
- Toolbar: toolbar
- ExecutorService: es
+ onNavigationItemSelected(MenuItem)
<<Interface>>
DeviceObserver
+ update(int type, String info)
Filter*
Filter, bTime, bDate
oup: rgDate, rgTime
: tvDateDisplay, tvTimeDsiplay
erFragment: df
erFragment: tf
RecordingsFragment*
- RecyclerView: rv
- RecordingListViewAdapter: lva
- File: storageDir
- LinearLayout: filter
- int: lastPosLoaded
- ImageView: waveform
- setupSearchOption(Menu m)
- setupDeleteAllOption(Menu m)
- setupDownloadAllOption(Menu m)
- setupFilterOption(Menu m)
- setupRefreshOption(Menu m)
HomeFragment*
+ Button: deviceSettings, recordings, instructions
+ TextView: deviceStatus
- int: deviceStatusText
update(int type, String info)
DeviceSettingsFragment*
- static final String[]: VALID_MODES
- TextView: battery, status
- Button: connect, sync
- Spinner: mode
- int: deviceStatusText
- DeviceController: dc
+ update(int type, String info)
RecordingListViewAdapter
- List<Recording>: dataList
- List<Recording>: dataListCopy
- File: storageDir
~ ListMediaPlayer: lmp
- ImageView: waveform
+ int: getItemCount()
+ addRecordingsToView(ArrayList<Recording> rs)
+ removeRecordingByID(String id)
+ removeAllRecordings()
+ refresh(ArrayList<Recording> rs)
+ searchLoadedList(String text)
+ sortByDateTimeAsc()
+ sortByDatetimeDesc()
Use
Use
JSONHandler
+ static final int: ON, AT, BEFORE, AFTER
- File: storageDir
- JSONObject json
- JSONObject: loadJSON()
+ removeRecording(String id)
+ deleteAll()
+ getMostRecent(int num)
+ getMoreData(int start, int end)
+ addRecording(String filename)
+ addTag(String id, Tag t)
+ addTags(String id, ArrayList<Tag> ts)
+ deleteTag(String id, Tag t)
+ ArrayList<Recording>: searchJSON(S
text)
+ ArrayList<Recording>:
filterJSON(LocalDate d, int dFilterType,
LocalTime t, int tFilterType)
+ int: getNumberof Recordings()
FileHandler
- File: storageDir
+ deleteFile(String filename)
+ saveFile(InputStream in, String filename)
Recording
- String: filename
- LocalDateTime: dt
- String: id
- ArrayList<Tag>: tags
1+ String: getFilePath()
+ LocalDateTime: getDateTime()
+ String: getID()
+ ArrayList<Tag>: getTags
+ addTag(Tag t)
+ removeTag(Tag t)
Created in HomePage and passed to classes that need it so that they all use the same Object (also storageDir)

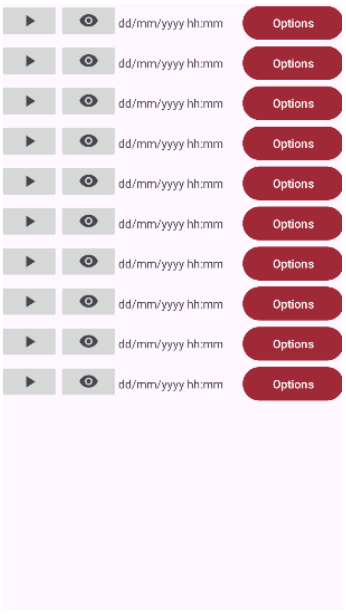

<<Interface>>
ModelCallback
+ onComplete(ArrayList<String> result)
ModelHandler
- Executor: e
- Python: py
- File: storageDir
- Model: m
+ processRecording(String filename, ModelCallback cb)
- createSpectromgram(String filename, int length)
- ByteBuffer: preprocessing()
- int: predict(ByteBuffer input)
Use
DeviceController
- int: dataStatus
- boolean: connected
- boolean: writeBusy
- int: mode
- ByteArrayOutputStream: baos
- ArrayList<DeviceObserver>: os
- checkPermissions()
+ int: getBattery()
+ setMode(String newMode)
+ boolean: isConnected()
+ connect()
+ disconnect()
+ sync()
+ attach(DeviceObserver o)
+ detach(DeviceObserver o)
+ notifyObservers(int type, String info)
<<Interface>>
DeviceSubject
+ static int: CONNECTED
+ static int: DISCONNECTED
+ static int: NEW_DATA
+ attach(DeviceObserver o)
+ detach(DeviceObserver o)
+ notifyObservers(int type, String info)

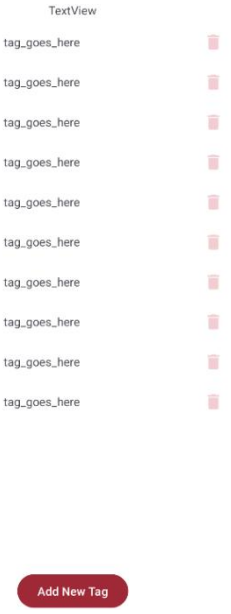
B. XML VIEWS

activity_home_page.xml: fragement_device_settings.xml: fragement_home.xml:



fragment_recordings.xml: manage_tags_view.xml:



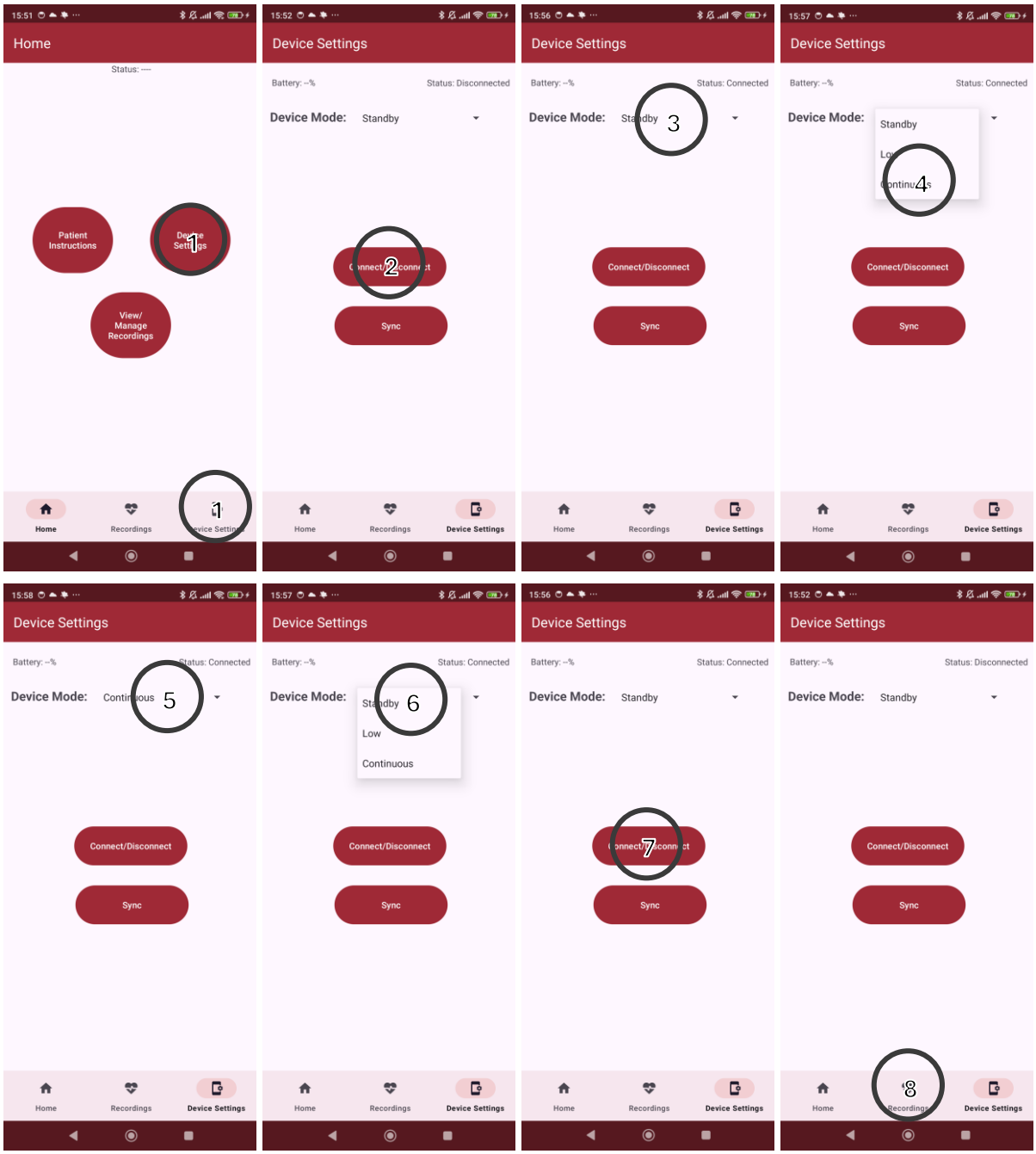


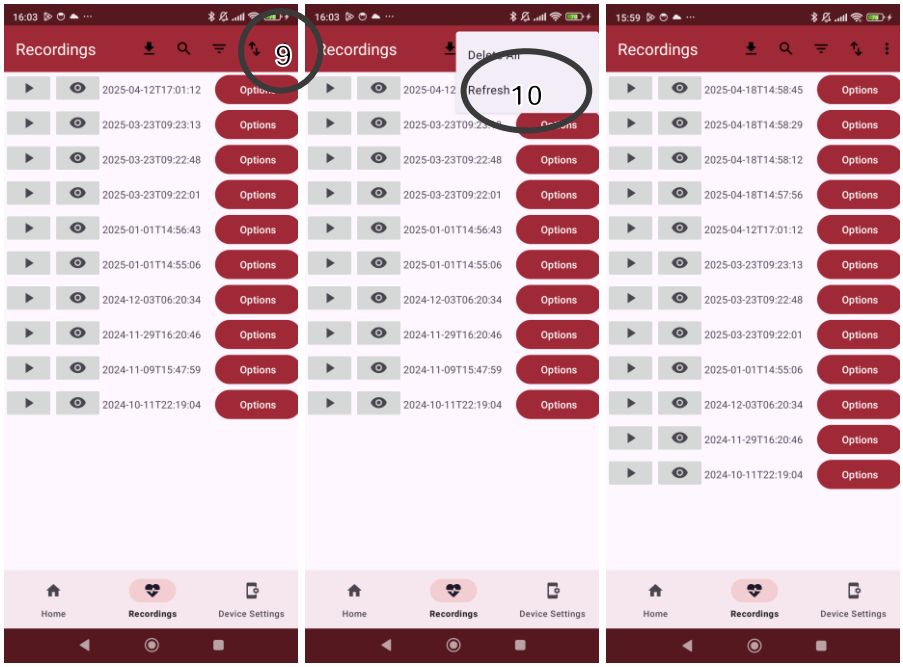
recording_item_loading.xml: recording_view_item.xml:

tag_app_added.xml: tag_user_added.xml

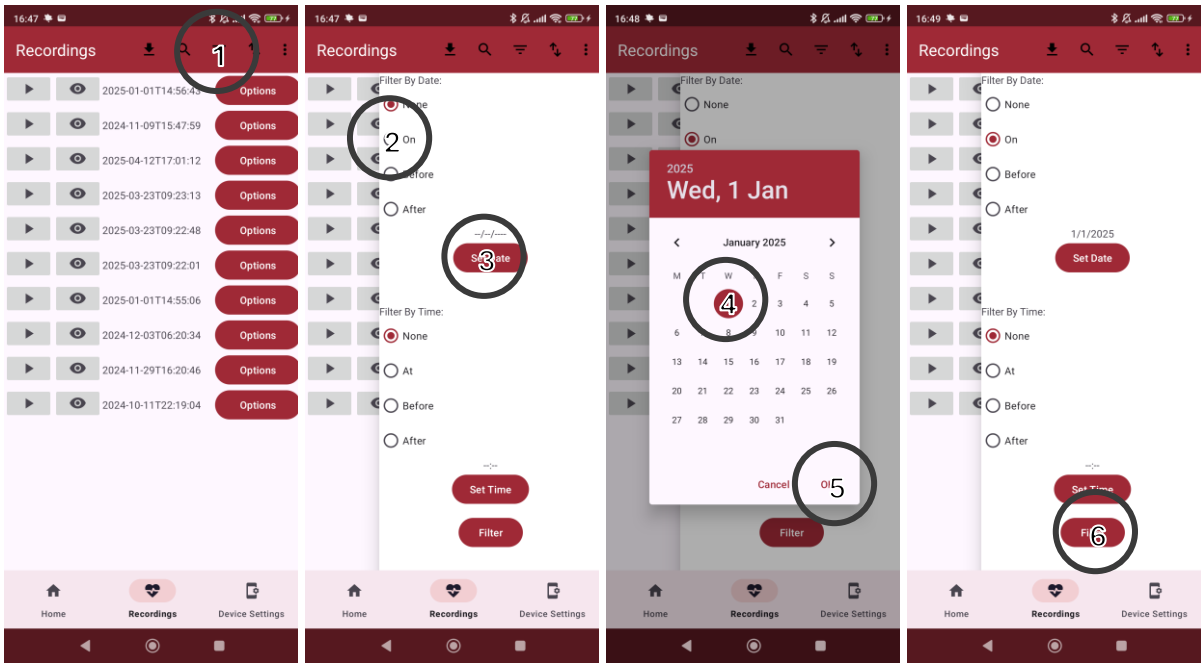
C. Full App Interaction Walkthroughs

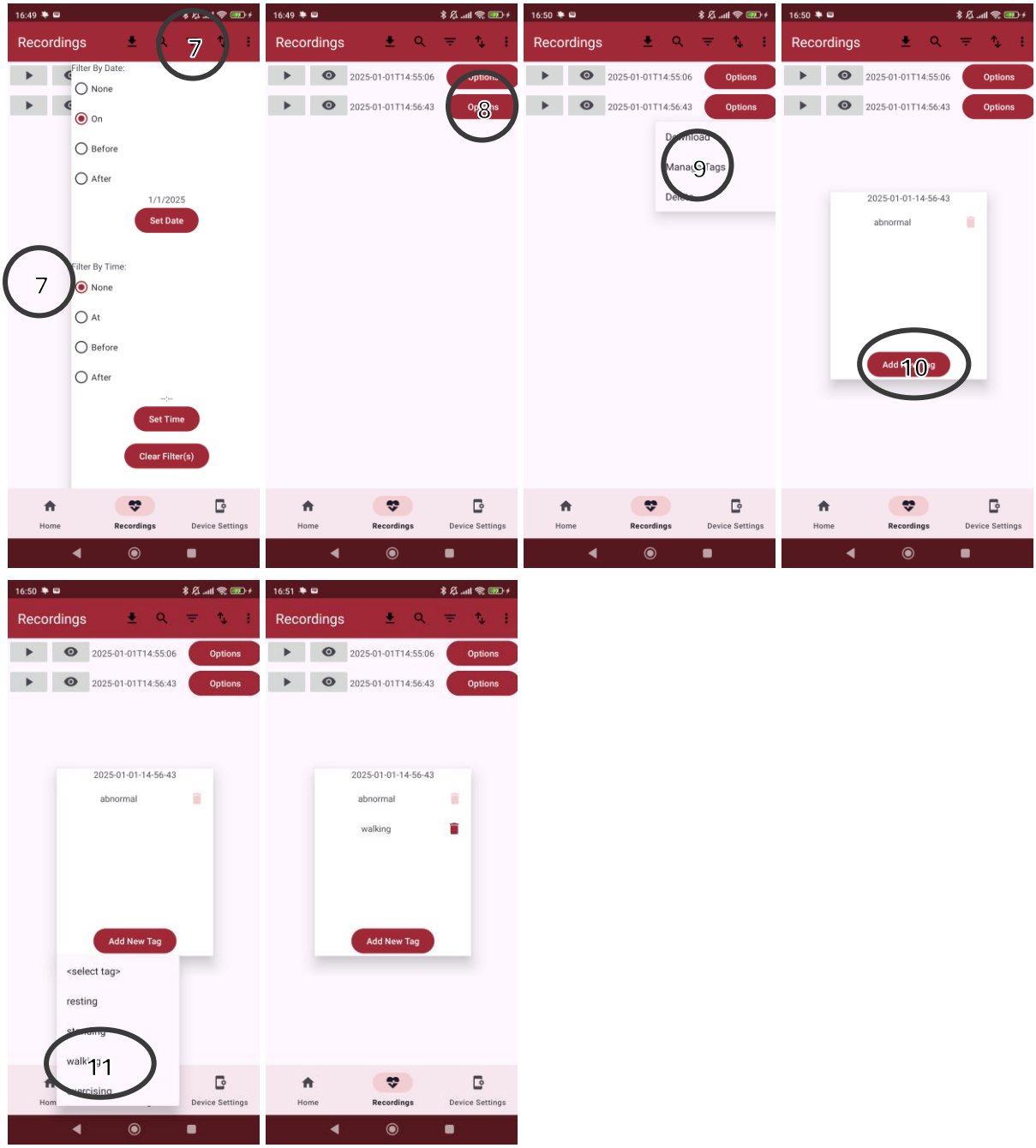
Connect, Record, Disconnect:



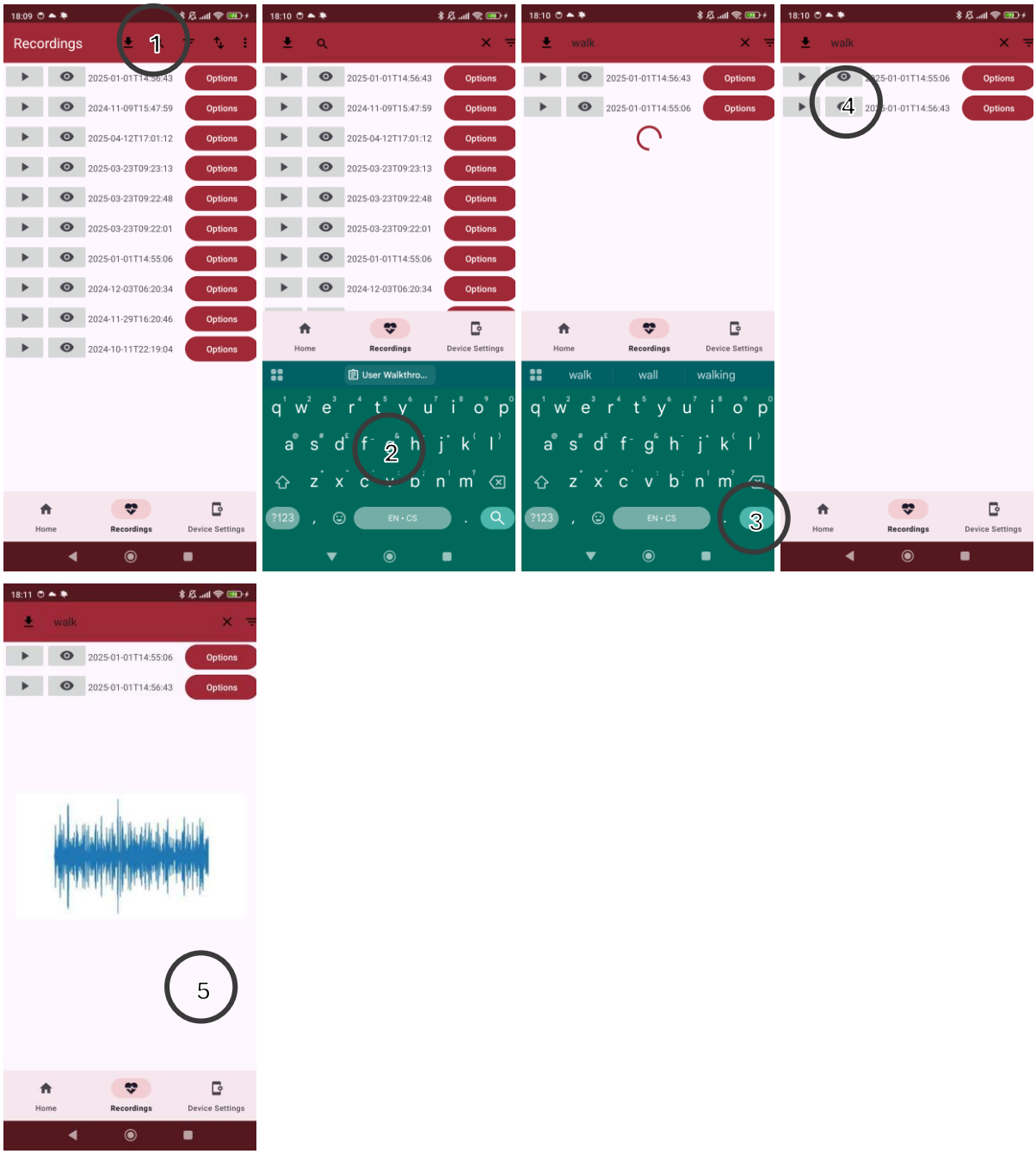


Filter, Add Tag:





Searching, Viewing Waveform:


Recordings
walk
Options
Home
Recordings
Device Settings
1
2
3
4
5

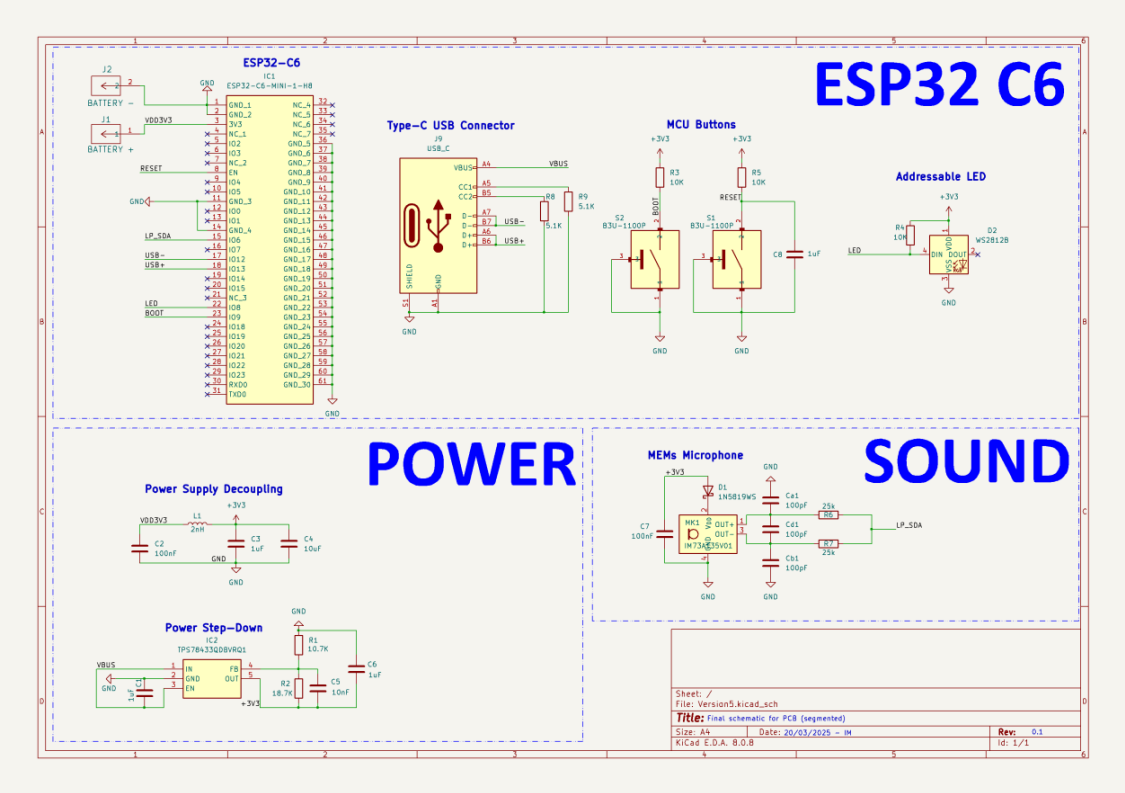
D. PRE-PROCESSING CODE (PYTHON)

```
# DESCRIPTION:
# Files for app integration

import librosa
import librosa.display
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt
import soundfile as sf
from PIL import Image

# Extend or trim signal
def extend_signal(signal, fs, target_duration):
    target_samples = int(target_duration * fs)
    original_samples = len(signal)

    if original_samples < target_samples:
        # Repeat if too short
             extended  =  np.tile(signal,  int(np.ceil(target_samples  /
original_samples)))
        return extended[:target_samples]
    else:
        # Trim if too long
        return signal[:target_samples]

# Basic bandpass filter (20-400 Hz)
def bandpass_filter(audio, sr, lowcut=20, highcut=400, order=4):
    nyquist = 0.5 * sr
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = signal.butter(order, [low, high], btype='band')
    return signal.filtfilt(b, a, audio)

# Generate and save spectrogram
def plot_mel_spectrogram(input_wav, output_img):
    y, sr = librosa.load(input_wav, sr=None)
    y_filtered = bandpass_filter(y, sr)

       mel_spec  =  librosa.feature.melspectrogram(y=y_filtered,  sr=sr,
n_mels=128, fmax=800)
    mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)

    plt.figure()
    librosa.display.specshow(mel_spec_db, sr=sr)
    plt.axis('off')  # don't want axes in the image
    plt.savefig(output_img, bbox_inches='tight', pad_inches=0)
    plt.close()

def plot_waveform(input_wav, output_img):
    y, sr = librosa.load(input_wav, sr=None)

    # plot waveform image
    plt.figure()
    librosa.display.waveshow(y, sr=sr)
    plt.axis('off')  # don't want axes in the image
    plt.savefig(output_img, bbox_inches='tight', pad_inches=0)
    plt.close()
```

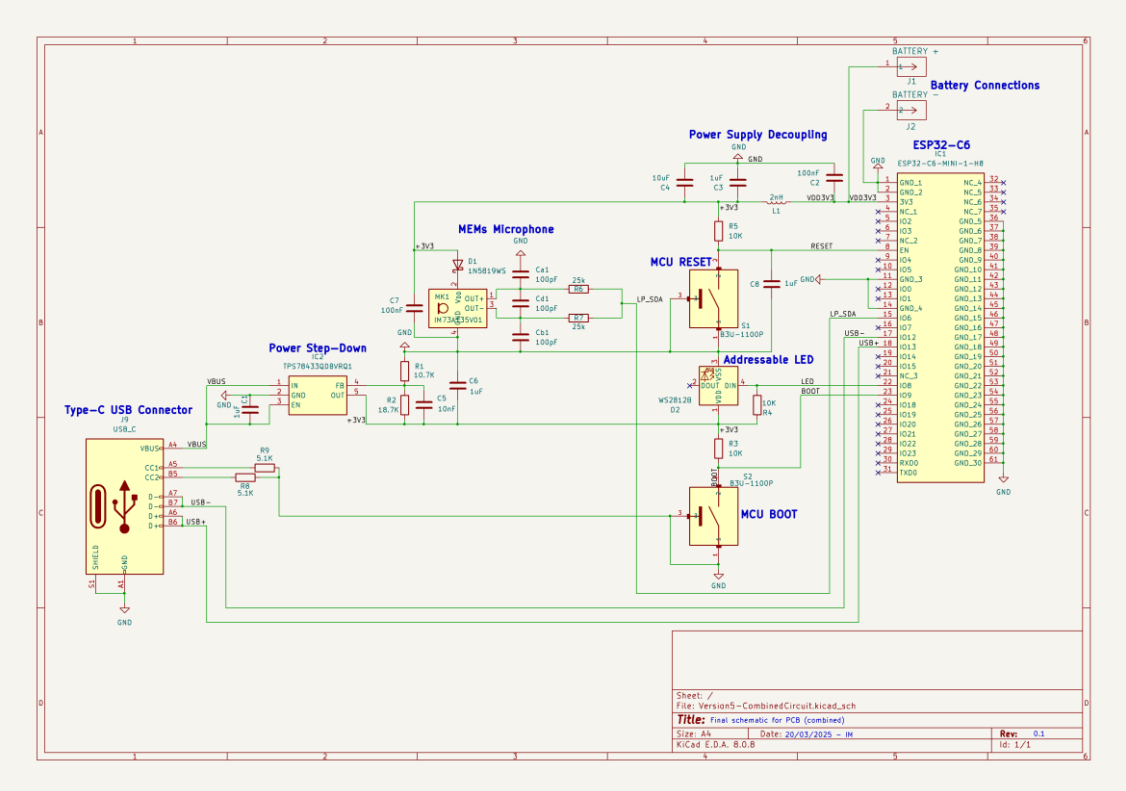
```
    # compress
    image = Image.open(output_img)
    width, height = image.size
    new_size = (width//2, height//2)
    resized_image = image.resize(new_size)
    resized_image.save(output_img, optimize=True, quality=50)

# Main function
def main(target_duration, input_wav, extended_wav, mel_spectrogram_img,
waveform_img):

    # Load with librosa
    y, sr = librosa.load(input_wav, sr=None)
    y_extended = extend_signal(y, sr, target_duration)
    sf.write(extended_wav, y_extended, sr)  # save extended aduio
    plot_mel_spectrogram(extended_wav, mel_spectrogram_img)
    plot_waveform(input_wav, waveform_img)

    print("COMPLETE")
```

E. Circuit Schematics

Segmented view:



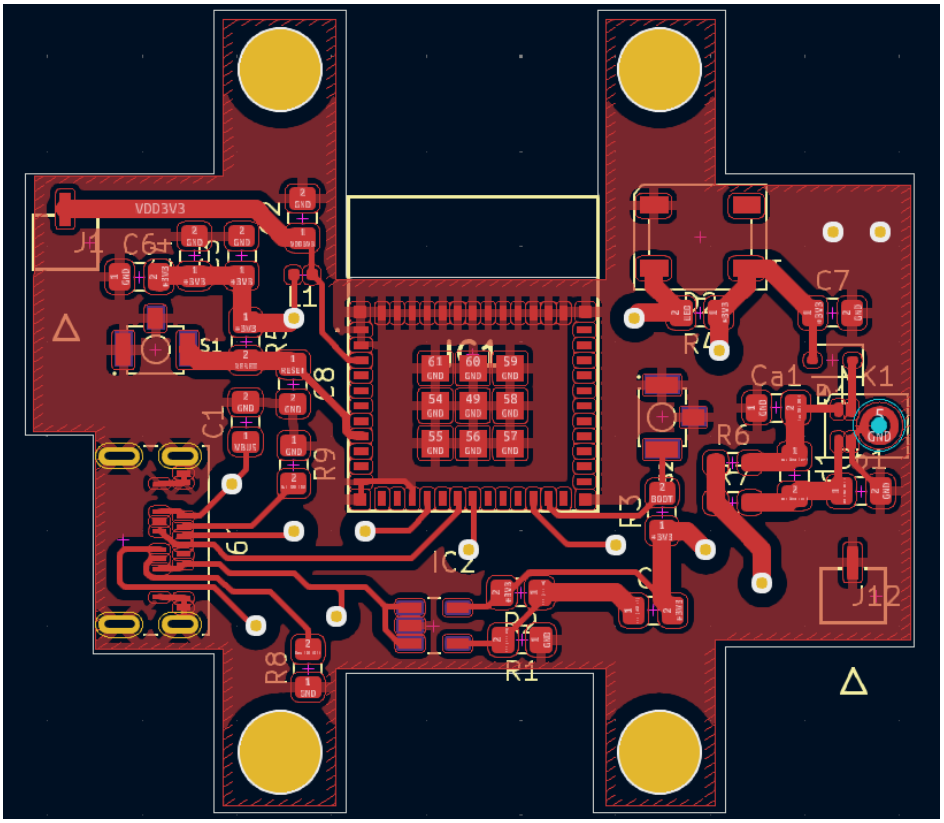
Combined view:



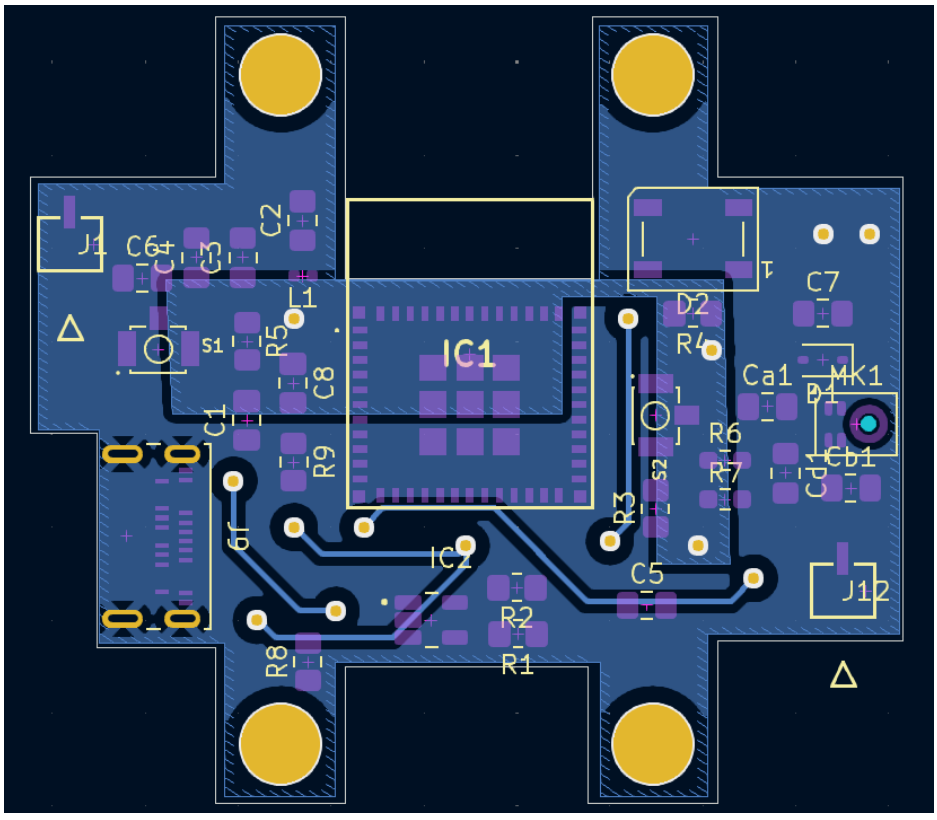
F. PCB BILL OF MATERIALS

G. PCB LAYOUT SCHEMATICS

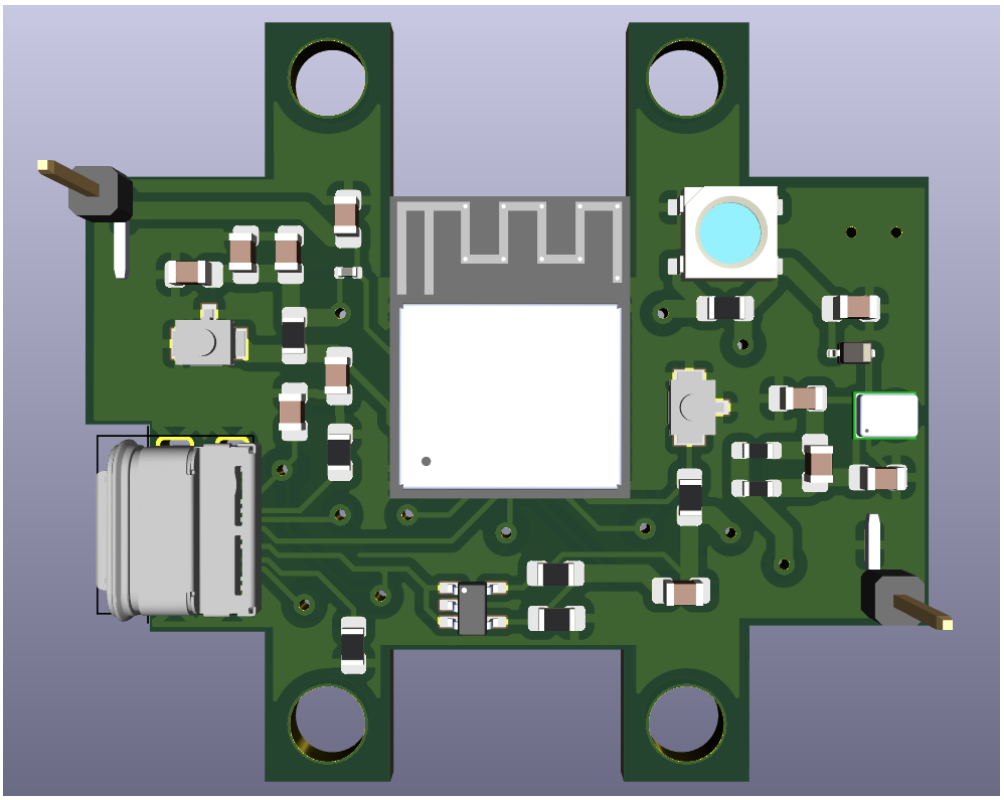
Top plane:

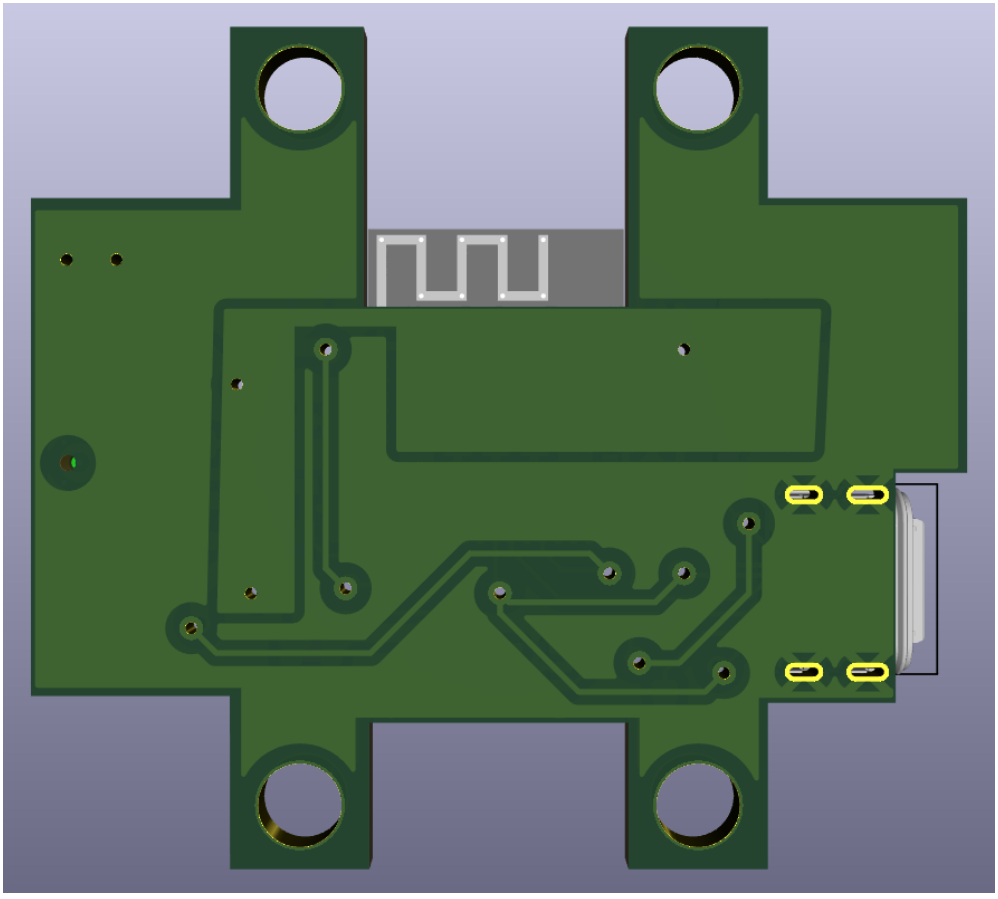


Bottom plane:



3D render of final version (Version 4):





H. COMPLETE BILL OF MATERIALS