Adaptive and Heuristic Artificial Intelligence (AI) IoT Edge for Providing Proactive Healthcare to Rural and High-Risk Patients

A Project Report

Presented to

The Faculty of the College of

Engineering

San Jose State University

In Fulfillment

Of the Requirements for the Degree

Master of Science in Computer Engineering

Master of Science in Software Engineering

By

Priyanshi Jajoo

Parvathy Kannankumarath Madom Krishnan

Harshraj Mahesh

Thasleem Banu Tajudeen

December 2020

APPROVED Chandrasekar Vuppalapti

11/30/2020

Professor Chandrasekar Vuppalapati, Project Advisor

Professor David Bruck, Director, MS Computer Engineering

Professor Dan Harkey, Director, MS Software Engineering

Rod Fatoohi, Department Chair

ABSTRACT

Adaptive and Heuristic Artificial Intelligence (AI) IoT Edge for Providing Proactive Healthcare to Rural and High-Risk Patients

By

Priyanshi Jajoo, Parvathy Kannankumarath Madom Krishnan, Harshraj Mahesh, Thasleem Banu Tajudeen

Accessible healthcare is crucial for the overall well being of society. One of the areas that need to be focused on is rural residents. Rural communities often suffer without adequate medical care because of the lack of healthcare services and providers within reach. This leads to disease progression and higher treatment costs. For instance, the prevalence of respiratory illnesses such as chronic obstructive pulmonary disease (COPD) is high among rural areas compared with people who live close to the cities [1]. Care for high-risk patients is another area that needs to be addressed. Patients with serious pre-existing medical conditions need to be monitored regularly to take timely measures. Senior citizen falls are one such example. According to the WHO, among older people over 64 years of age, 28-35% fall each year [2]. Falls are the primary cause of injuries for older people affecting them physically and psychologically. Recent studies have shown that AI-enabled technology can eliminate the physical distance between the clinicians and patients and also transform the way diseases are being diagnosed and treated.

Today, many of the AI-assisted healthcare devices are designed for healthcare professional's use and they are operated on cloud storage. Cloud-based services cause latency for applications such as fall detection that require real-time predictions. So, it is important to build an edge operated, reliable, and easy to use AI healthcare system for patients.

In this project, as an effort to mitigate the inequality between the urban and rural healthcare systems, an adaptive heuristic AI system is developed. The system acquires data from patients using sensors. The collected data is processed and analyzed in an edge device for faster prediction. Then, the system proactively notifies caregivers or doctors if an illness is diagnosed.

Acknowledgments

The authors are deeply indebted to Professor Chandrasekar Vuppalapati for his invaluable comments and assistance in the preparation of this study

Table of Contents

List of Figures

List of Tables

Chapter 1. Project Overview

1.1 Introduction

The healthcare systems are overburdened, and a countless number of patients are deprived of treatments at the right time. One of the most crucial factors for this is accessibility to healthcare. For instance, in rural areas or underserved communities, patients often travel long distances to visit a clinic. Another example is when patients with critical illnesses require fast access to healthcare facilities.

There are myriad possibilities for AI technologies to be used in healthcare, starting from at-home diagnosis to preventive treatments. AI can also be used to increase accessibility to healthcare and suggest proactive treatment methods for health issues. This can be done by creating a solution that is cost-effective, easily available, portable, and most importantly reliable. Several factors contribute to this being effective. In the case of patients with little or no access to doctors and clinics, an AI technology-based solution that requires minimum equipment and human intervention serves best, whereas for high-risk patients quick response is vital.

The proposed solution consisting of a mobile and web application aims to provide proactive healthcare to rural and high-risk patients. Patients take health tests using the mobile application that serves as an IoT device. It provides fast and reliable diagnosis and treatment recommendations after analyzing the data collected from the patient. It is not constrained by network connectivity. It also notifies primary contacts upon identifying a serious health issue. The web application serves as a dashboard that is used to view the history of health tests and its recommendations. The system provides recommendations for respiratory issues and falls. Respiratory issues are analyzed by processing lung sounds recorded using a stethoscope and detecting crackles and wheezes in them. Falls are analyzed by checking the data from sensors on the mobile phone.

1.2 Proposed Areas of Study and Academic Contribution

The potential of AI has touched every field, and the healthcare industry is no exception. There are numerous applications of AI in healthcare. Many of these applications focus on aiding patients who are more prone to certain illnesses or living in rural areas where there are no adequate medical care facilities. Detecting respiratory-related diseases using AI technologies is one such example. Respiratory illness is the third leading cause of death around the world, and deaths caused by diseases like asthma and chronic obstructive pulmonary disease (COPD) are higher in rural areas because of the lack of health programs that offer early diagnosis and treatments. The annual cost of COPD in the United States is estimated to be $49 billion [3]. Senior citizens' fall is another major issue that needs to be addressed. According to the CDC, more than 3 million people are treated for fall-related injuries and more than $750 million is spent on fatal fall injuries every year [3]. An adaptive and heuristic AI system that operates on the edge level with the ability to prevent or diagnose these conditions by the early stage will improve prognosis and give better results to the treatments. So, there is a dire need to build an IoT edge system powered with AI.

The adaptive nature of the AI system ensures that the model updates itself based on the feedback from the environment and makes accurate predictions on the new data. Running inference on the edge device makes it possible for the patients to use the applications even without network connectivity. Apart from notifying the user or caregiver, the AI-powered IoT edge lets the patient monitor their health regularly at home and share the findings with their healthcare providers whenever required. This kind of at-home health monitoring applications powered with AI can highly benefit high-risk patients and people residing in rural areas as it avoids unnecessary trips to the doctor's office.

The goal of this project is to build a mobile application with a machine learning model deployed in it, and focus on use cases like detection of crackles and wheezes, and falls. The data is acquired using IoT sensors or external devices and processed at the edge level to reduce the latency caused by going back and forth to the cloud server. The mobile application then runs inference on the data and notifies the caregiver and provides recommendations based on the

findings. The collected data and the user's history will be stored in the cloud for backup. A web application for the dashboard view is also proposed. If the model can predict abnormalities with high accuracies, the quality of life will surely improve and the country's financial burden will go down to a certain extent. Based on the performance, the mobile application can be further enhanced by including other healthcare issues that can be solved with AI.

1.3 Current State of the Art

IoT edge devices enabled with AI are generally designed to provide innovative healthcare solutions. From disease diagnosis to treatment recommendations, AI and IoT technologies provide a structured framework to ease the work of the medical personnel. Fall detection, respiratory sound analysis, diabetes detection, and stroke detection are some categories where AI-enabled IoT edge can provide effective solutions. This project involves the development of AI-enabled IoT edge for high-risk and rural patients. Several researchers have been evaluating the potential of AI and IoT in healthcare. Thomas Davenport and Ravi Kalakota [4] proposed that AI can perform equally or better than humans at key healthcare tasks, such as diagnosing diseases. The paper discussed categories of healthcare where AI can be used, like patient engagement and adherence, treatment recommendations, etc. Additionally, for diagnosis and treatment recommendations, the paper discussed IBM's Watson tool, which employs machine learning algorithms and natural language processing capabilities to diagnose the type of cancer, whereas patient engagement and adherence expect proactive participation of patients, and big data analytics and AI are used to address these factors.

Jonathan Guo and Bin Li [5], researched AI solutions to bridge the gap between rural and urban healthcare in developing countries. They implemented AI in clinical decisions, electronic health records, diagnosis, personalized medicine, healthcare system management, and medical robots. This paper also suggested a multi-level medical AI service network for rural areas.

Higinio Mora et al.[6] proposed an IoT based computational framework for healthcare monitoring. The main idea was to fetch data from sensors or wearables like a smartwatch, blood pressure sensors, and inertia sensors, and then data was processed to get the best knowledge from

it. The proposed solution was robust to use in the mobile environment. They presented a case study related to monitoring the heart rate of footballers to illustrate their work.

SunGil Yoo and Dongik Oh [7] proposed a method to detect falls of elderly people. According to their research, hip joints damage is the most serious injury with severe consequences. They proposed a system that detects falls from the acceleration sensor attached to the wrist of the elderly people and they used artificial neural network-based deep learning methods to analyze falls.

Many research articles define methods to extract features from audio data using data science algorithms. Archana Ramalingam et al. [8] suggested methods to extract the most suitable acoustic parameters. Mel scaled spectrograms and Mel frequency cepstral coefficient (MFCC) components were used to train various neural networks like plain neural networks(NN), convolutional neural networks(CNN), and recurrent neural networks(RNN).

Morten Gronnesby et al. [9] offered a machine learning method to detect crackles in lung sounds recorded using a stethoscope. The analysis pipeline of their work was based on the features extracted from audio files. Several classifiers were evaluated and SVM with a radial basis function kernel proved to be the best for the detection of crackles.

AI and IoT edge technologies have shown incredible efficiency in the healthcare industry. This project involves the development of an android application and web application, which is AI-enabled and can detect crackles, wheezes, and falls.

Remote monitoring of a patient can be done by using IoT edge which in this case is a smartphone. In this project, a stetho-microphone is used to record respiratory sound in a smartphone. An in-built gyroscope sensor is used to track fall. Data collected is processed on the smartphone. From collected data, features are extracted [10]. Machine learning algorithms like neural networks, support vector machines(SVM), and k-nearest neighbors (KNN) are used to train and test the data.

Healthcare-related smartphone applications have transformed many aspects of the medical sector. This project proposes an android mobile application process that uses recorded

respiratory sound and sensor data for falls using AI algorithms and enables patients to track their health. A web application, that provides a dashboard, is used to display the features of the respiratory sound to its user and possible fall situations. It provides the patient with a history of health tests and its recommendations. A notification system is also designed to provide notifications to patients' primary contacts.

Chapter 2. Project Architecture

2.1 Introduction

The project features a heuristic model that will be trained with a health-related dataset. Once the model is built, the model's weight is adjusted for precision. The model is built using Tensorflow. Once the model is built using Tensorflow it is converted to the TensorFlow Lite model. This Tensorflow Lite model is deployed in an Android application. This project also features a web dashboard that can be used by a user to track their health record. An architecture for the above-mentioned workflow has been illustrated below.

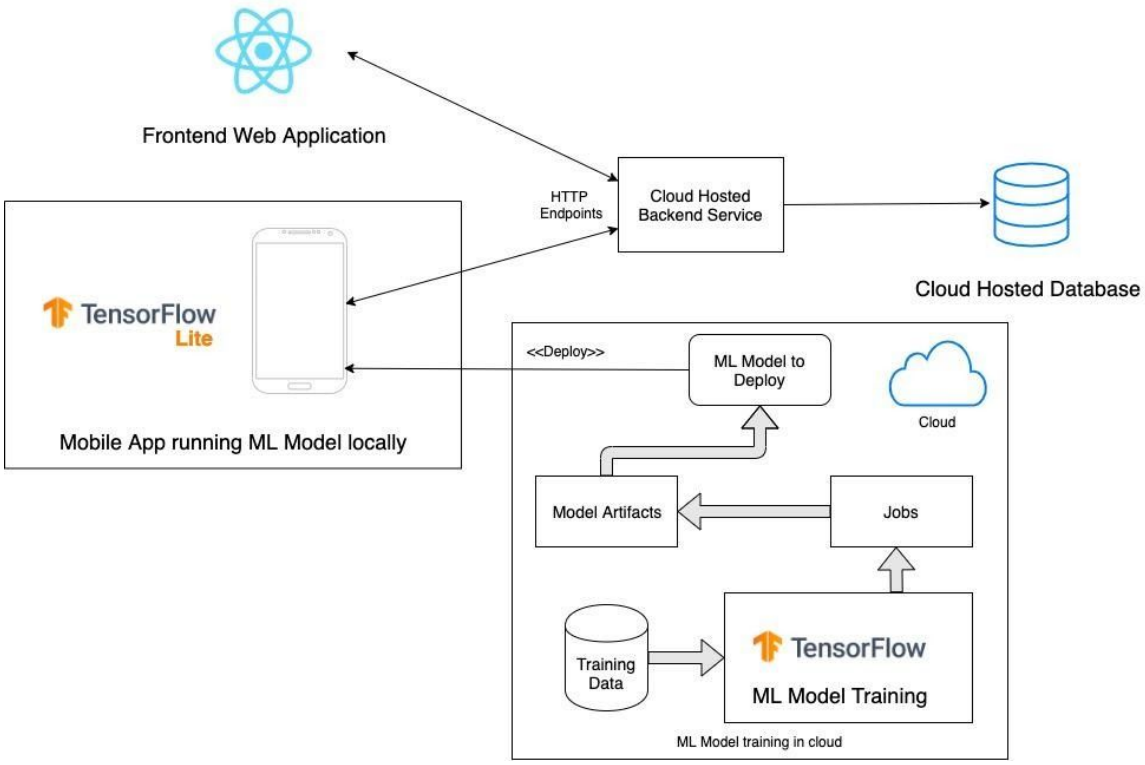


Figure 1: Architecture diagram

2.2 Architecture Subsystems

The system consists of 3 subsystems namely mobile application, web application, and building machine learning model.

2.2.1 Mobile Application

An android mobile application is built in which users can use a smartphone to track their respiratory health. The mobile app hosts a neural network locally to predict users' conditions and suggest required actions or precautions to be taken by users. The app is built using the native mobile platform, Android, and the neural network is hosted within the mobile application using the Tensorflow Lite model, that is generated after compressing an existing Tensorflow model.

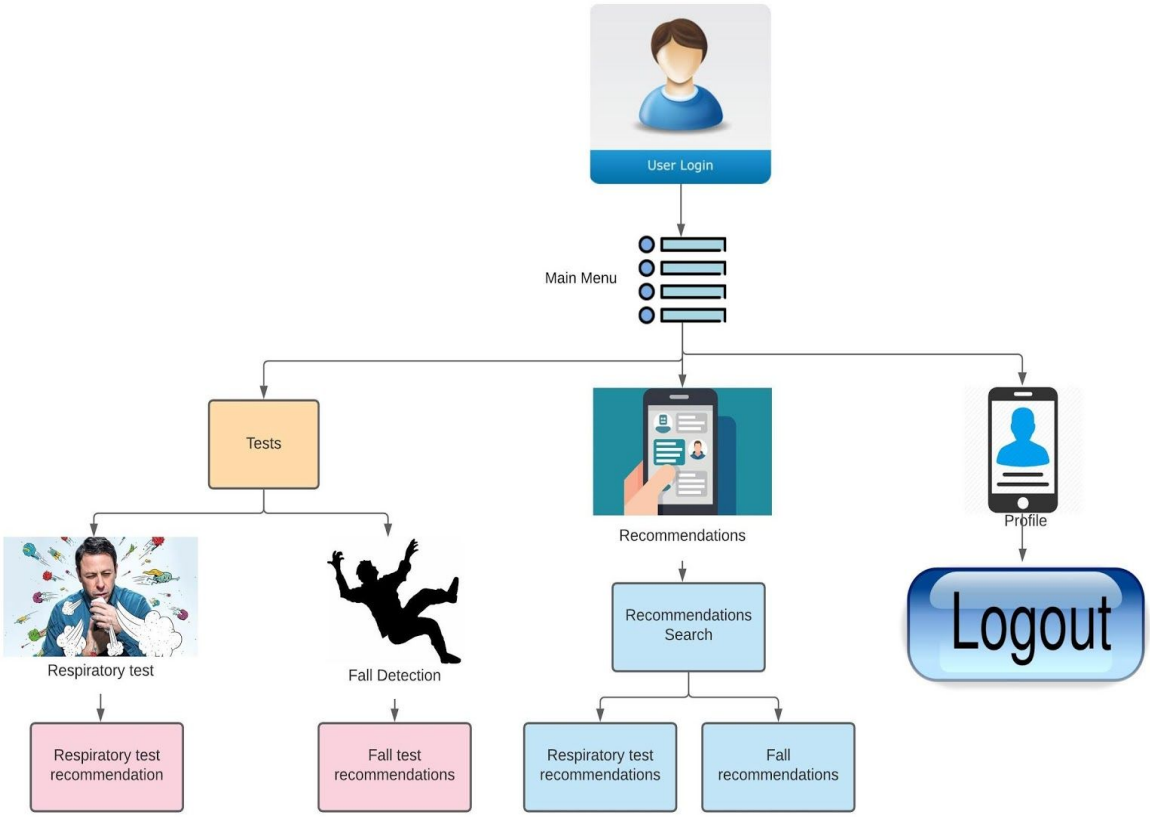


Figure 2: Pages hierarchy of the mobile application

The application will make API calls to the cloud-hosted backend server to save/update user data. Also, the application will learn from the user which makes the machine learning model adaptive.

2.2.2 Web Application

A single page application (SPA) is built using ReactJS, an open-source javascript library to build user interfaces. In this web application, the users can login and view their profile and dashboard. The dashboard contains the user's health-related statistics. The web application communicates with a backend server which will be hosted in the cloud in AWS. The backend server serves user requests coming from the web application with the help of a database. Firebase, a cloud-hosted managed NoSQL database is used. A NoSQL database is used as it is highly scalable horizontally and can have unstructured data.

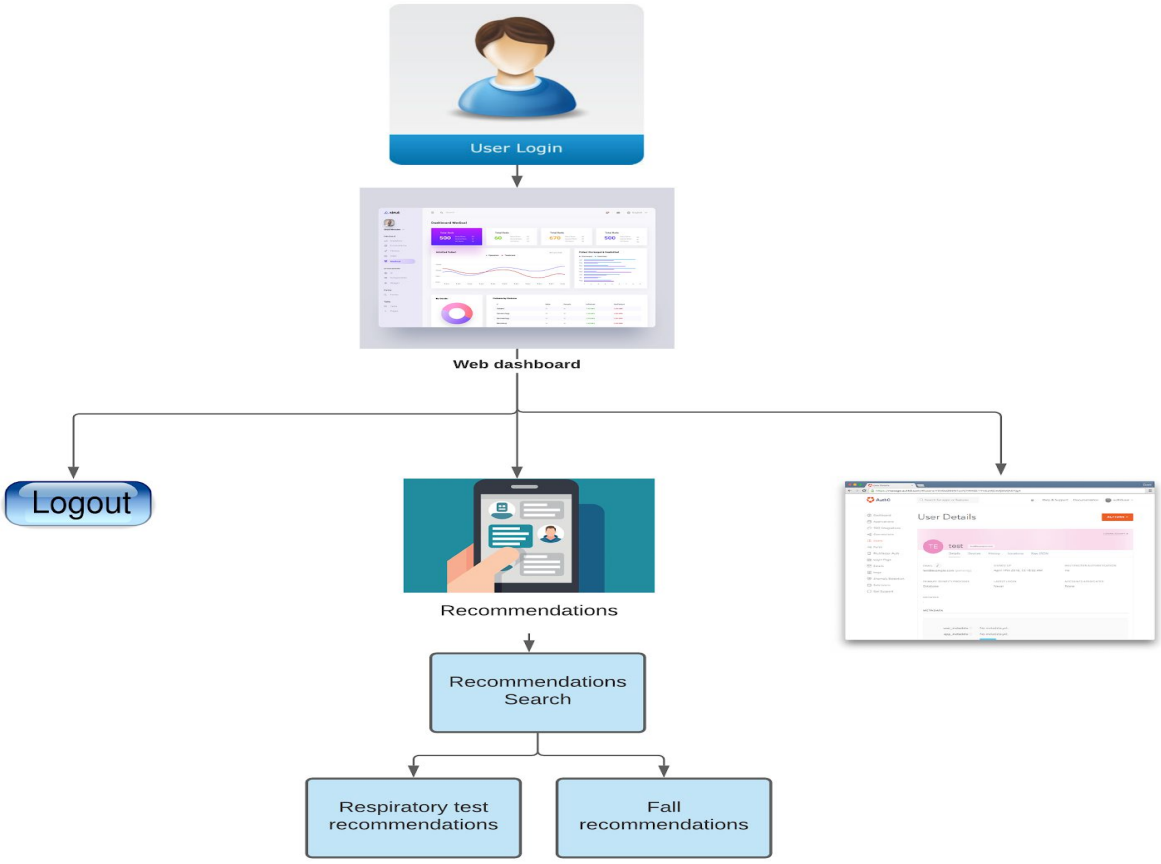


Figure 3: Pages hierarchy of web application

The web application is deployed in the cloud to maintain high scalability and availability in Firebase hosting. The ReactJS web application is built using HTML, CSS, and Bootstrap. Also, the web application is responsive so that it can also cater to users accessing the application using a smaller device like smartphones or tablets.

2.2.3 Building Machine Learning Model

Building an effective machine learning model that can be interpreted to make predictions on unseen data involves the following process.

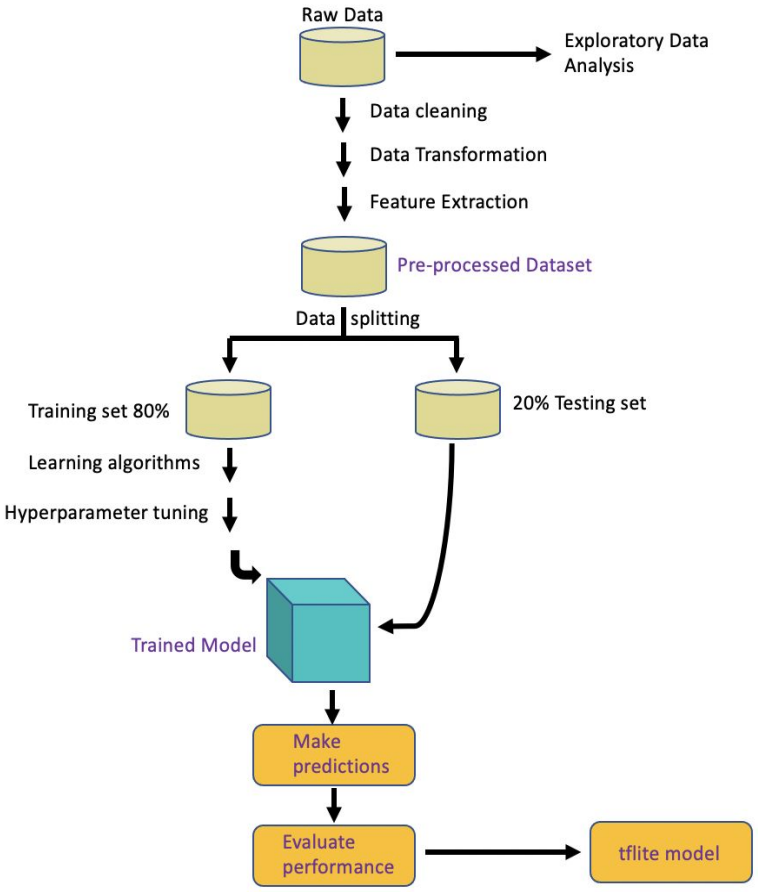


Figure 4: Machine learning model flow

Data Collection and Exploratory Analysis

Data collection is the process of gathering relevant data that can serve the purpose of the proposed architecture; i.e., prediction of abnormal lung sounds and detection of fall events. In this project, pre-collected data that is open to the public are selected for building a machine learning model. Exploratory data analysis helps to understand the collected data by finding patterns, outliers, and summary statistics. Making sense of the collected data is crucial for data preprocessing which involves data cleaning and feature extraction.

Data Preprocessing

In the preprocessing step the collected raw data are transformed in such a way that it can be fed to the neural network for training. This step typically involves data cleaning, data transformation, data reduction, and sampling.

Training & Testing the Model

To accomplish the proposed design two different machine learning algorithms are trained. One for predicting respiratory illness and another one for fall detection. In both the settings, the preprocessed data is split into training and testing sets with an 80/20 ratio.

Data is trained and tested using different machine learning algorithms and the one with high classification accuracy, high sensitivity, and specificity on the test data is chosen for deployment.

Tflite Model Deployment on Android Applications

The best models chosen for prediction are converted to tflite using a TensorFlow Lite converter and deployed in an Android application using Android Studio.

Chapter 3. Technology Descriptions

3.1 Client Technologies

3.1.1 React.js

React.js is the JavaScript library used for developing the UI for the single page web application. It is built using HTML5, CSS3, and Bootstrap. Setup for this includes installing and using libraries from npm like react, react-dom among others. In the web application, users can see their health records, previous tests, and recommendations. Dashboard, recommendations, and profile sections are developed for the same.

3.1.2 Android

Android, the native mobile platform by Google is used for developing the mobile application. Setup includes Android Studio and Android SDK. It is using the mobile application, the user takes health tests and medical records saved in the database. In the android application, machine models are deployed, to predict the health test results.

3.2 Data Analytics Technologies

3.2.1 Python

Python programming language is used in the machine learning section of the project. It is used for fall and respiratory datasets, exploratory analysis, data cleaning and preparation, developing and testing models, and creating TensorFlow Lite models.

3.2.2 Python Libraries

Several Python libraries are used in the machine learning section of the project. Librosa library is used for respiratory dataset audio files analysis. It is used for audio data files, loading, and the subsequent extraction of features. pyAudioAnalysis is another library used for respiratory audio analysis. NumPy library is used for array manipulation scenarios like resizing arrays before feeding into a layer of neural network. SciPy library is used for numerical analysis. Pandas library, built on top of the NumPy package is used mainly for DataFrame data structure.

3.2.3 Jupyter Notebook

Jupyter Notebooks are used for creating python code that is run and the output is also saved to the files. It can be easily used in a collaborative environment. This offers a better way to run and see code than python files as a single line of code can be run alone instead of running the whole py file. Jupyter Notebooks are created for fall and respiratory datasets analysis, data cleaning, data preparation, and developing models and testing them.

3.2.4 TensorFlow

TensorFlow is a software library for machine learning. Deep neural network models used for machine learning are built using TensorFlow for both fall and respiratory datasets. For the fall dataset, a sequential Keras model is built using TensorFlow. Models are also converted into TensorFlow lite models.

3.2.5 Chaquopy

Chaquopy is the Python SDK for Android. It is used in android application development for running python code directly from android. This is used in the context of respiratory dataset analysis to extract features from the audio data since android programming has some drawbacks.

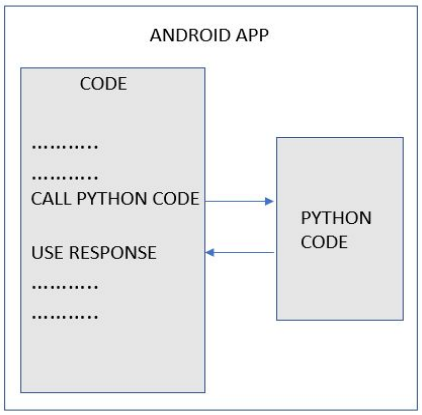


Figure 5: Chaquopy flow

3.3 Data-Tier Technologies

3.3.1 Firebase- Cloud Firestore

Cloud Firestore database by Firebase is used as the database for the project. When the user takes medical tests, the results are stored in this database. All the medical records and the analytics displayed in the web application dashboard is also fetched from this database. This database technology is used as it is easy to interact with both mobile and web applications.

Figure 6: Cloud Firestore

3.4 Deployment Technologies

3.4.1 Heroku

Heroku is the cloud application platform for the project. It is used to deploy and manage the SPA web application. It is deployed in Heroku using git. Git webhooks are set up, which automatically deploys the application in Heroku upon pushing into the remote git repository.

Figure 7: Heroku deployment

Chapter 4. Project Design

4.1 Client Design

4.1.1 UI Mockup

The flow of the web application is identified and a storyboard is created. The flow starts with the user logging into the web application. After a successful login, the user is taken to the dashboard view From here, the user can click on the view recommendation history button, to see recommendations to see both fall recommendations and respiratory test recommendations The user can also access his profile.

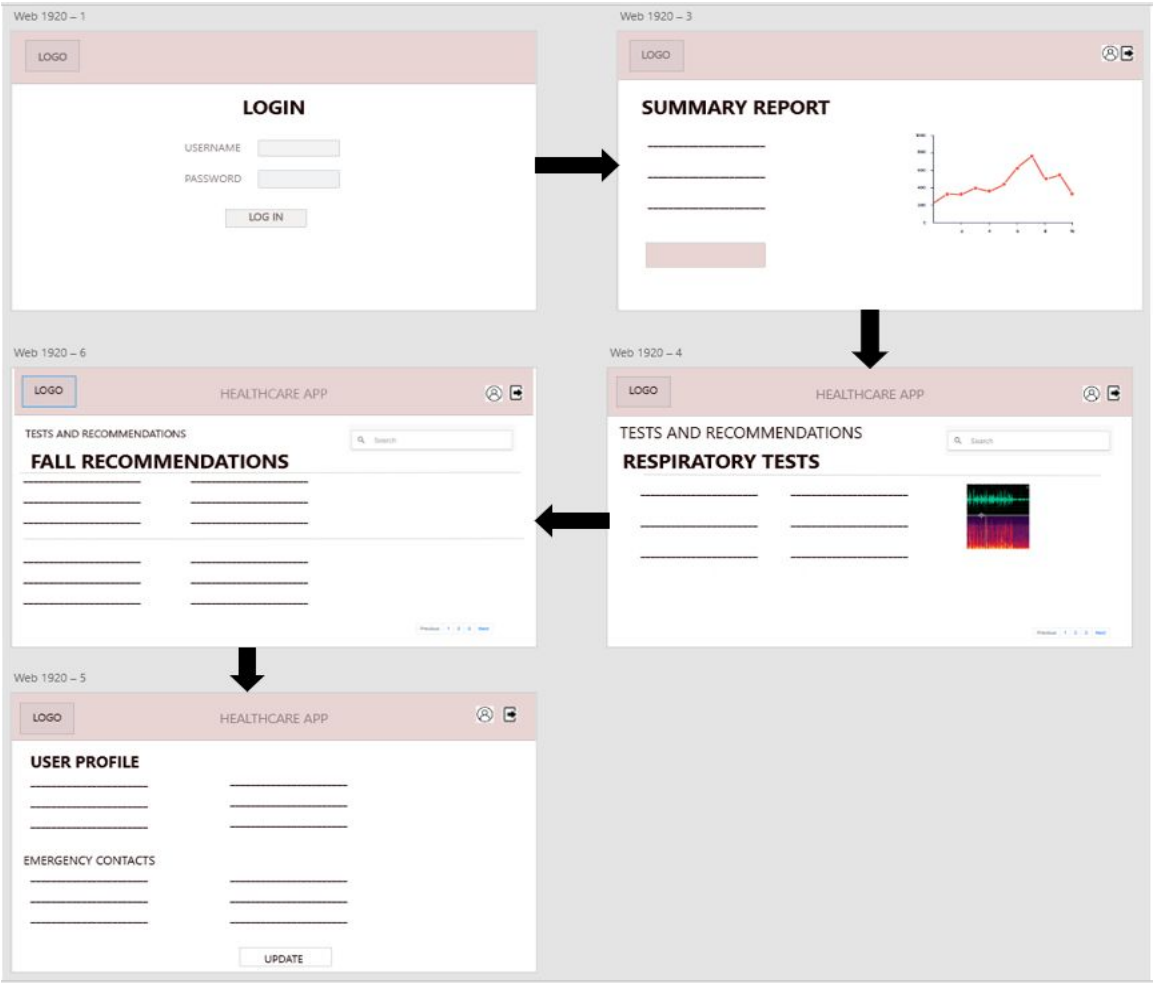


Figure 8: Web application storyboards

The flow of the mobile application is identified and a storyboard is created. The flow starts with the user logging into the mobile application. After a successful login, the user is taken to the main menu view which shows menu options for taking tests and viewing recommendations. The user can choose any test and provide data. After data is given, analysis is done and a recommendation for that is displayed. For a respiratory test, a page with instructions on how to record lung sounds is given. By following those steps, the user records the audio and then clicks on the analyze button. After analysis, the recommendation is displayed to the user. From the main menu, the user can click on the view recommendation history option, to see recommendations. Both fall recommendations and respiratory test recommendations are seen. There is also an option to search in the recommendations view. The user can also access his profile.

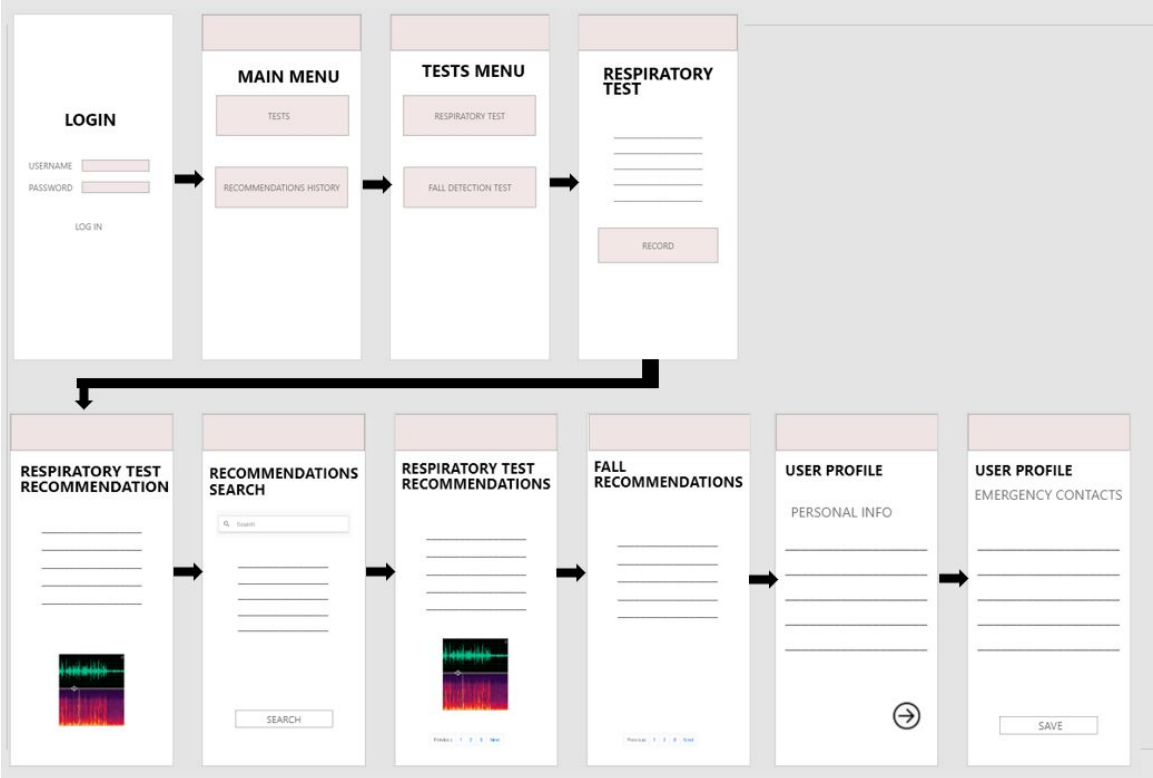


Figure 9: Mobile application storyboards

4.1.2 Wireframes

Wireframes for mobile application and web application is created using Adobe XD

Web Application

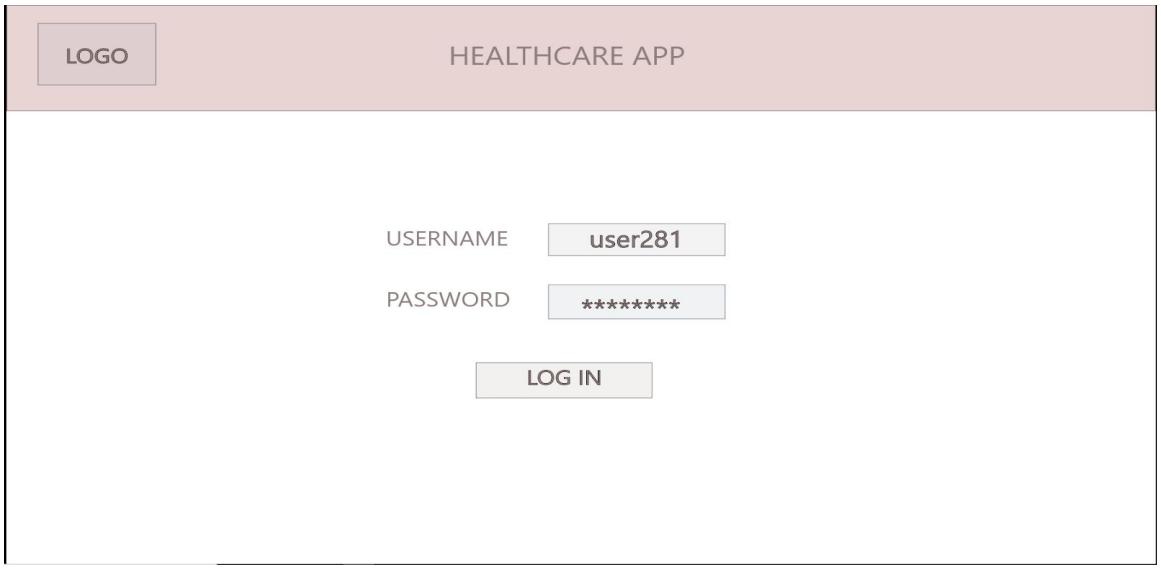


Figure 10: Login page

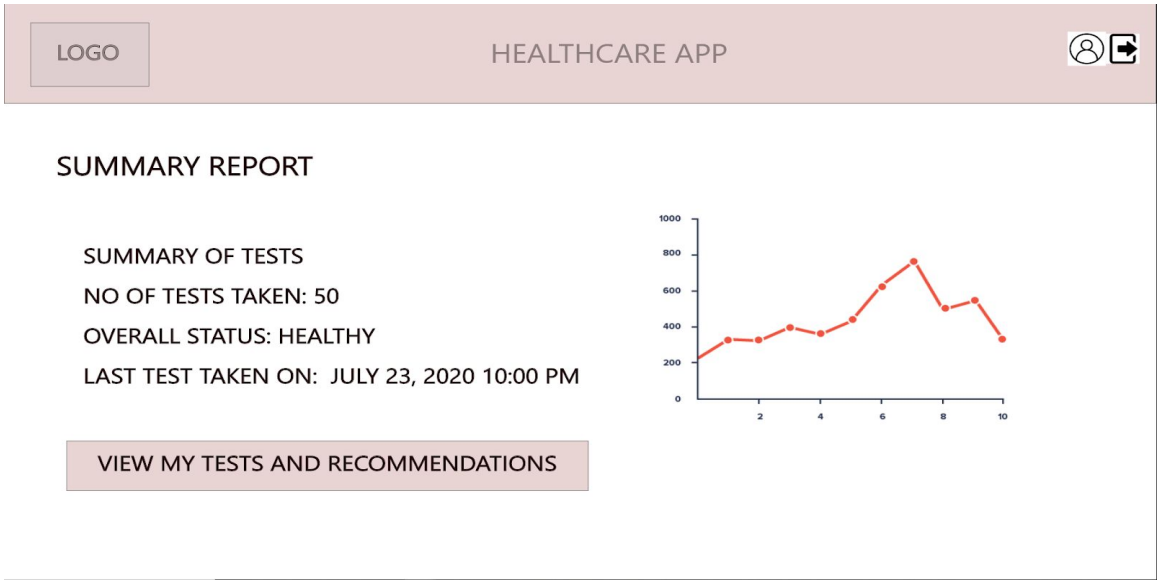


Figure 11: Dashboard page with a summary report

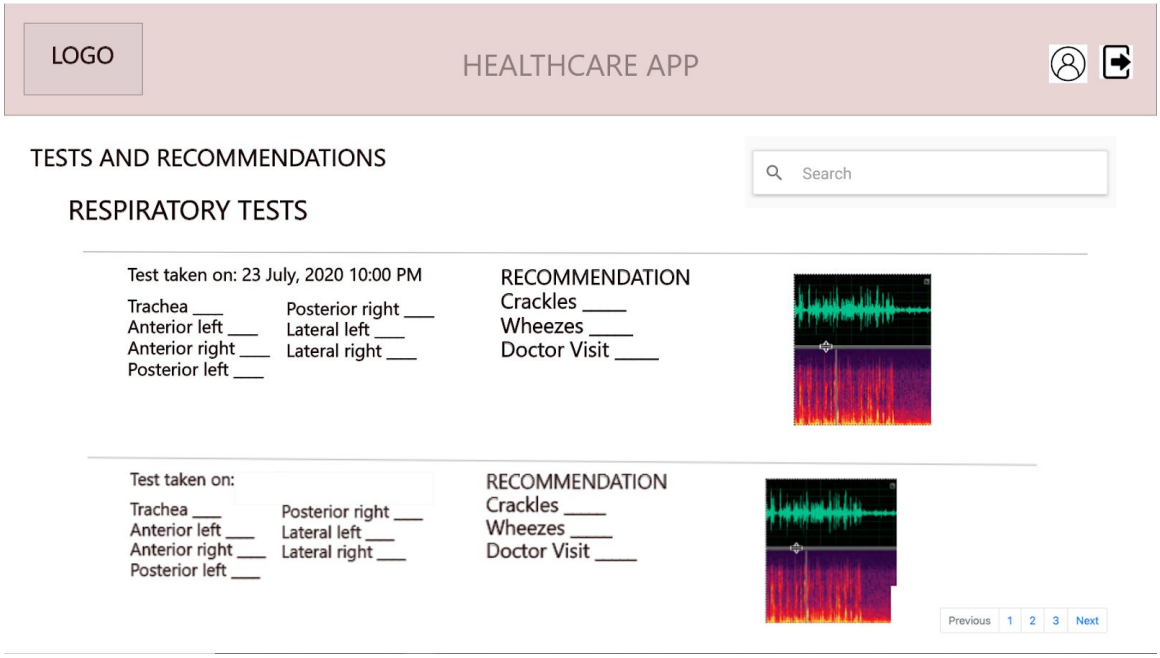


Figure 12: Recommendations page for respiratory tests

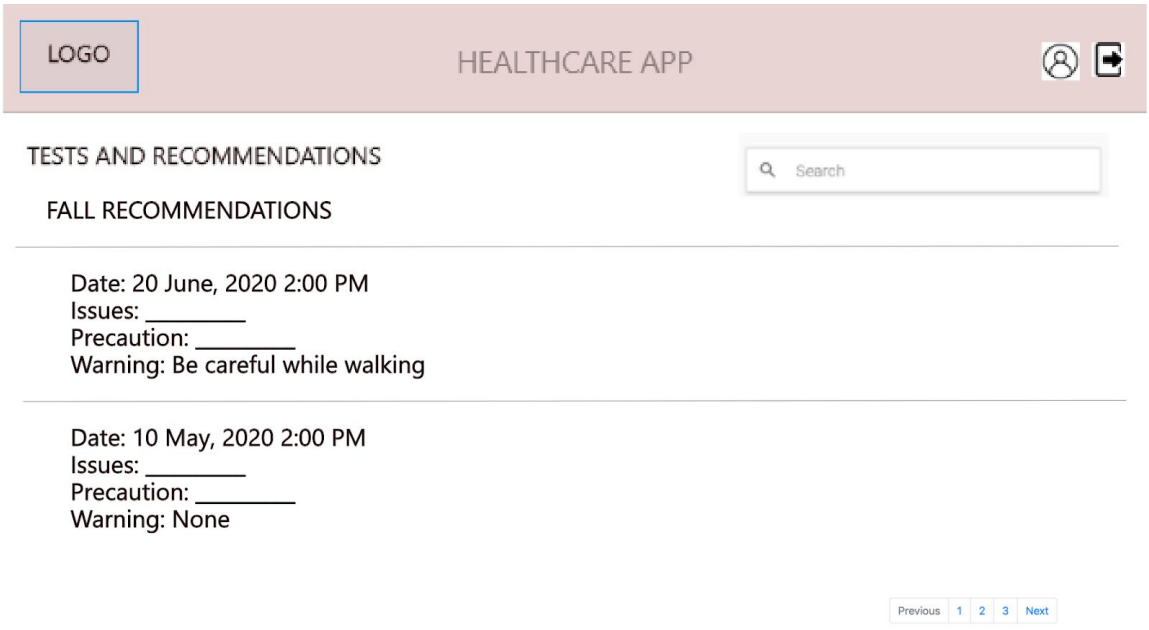


Figure 13: Recommendations page for fall

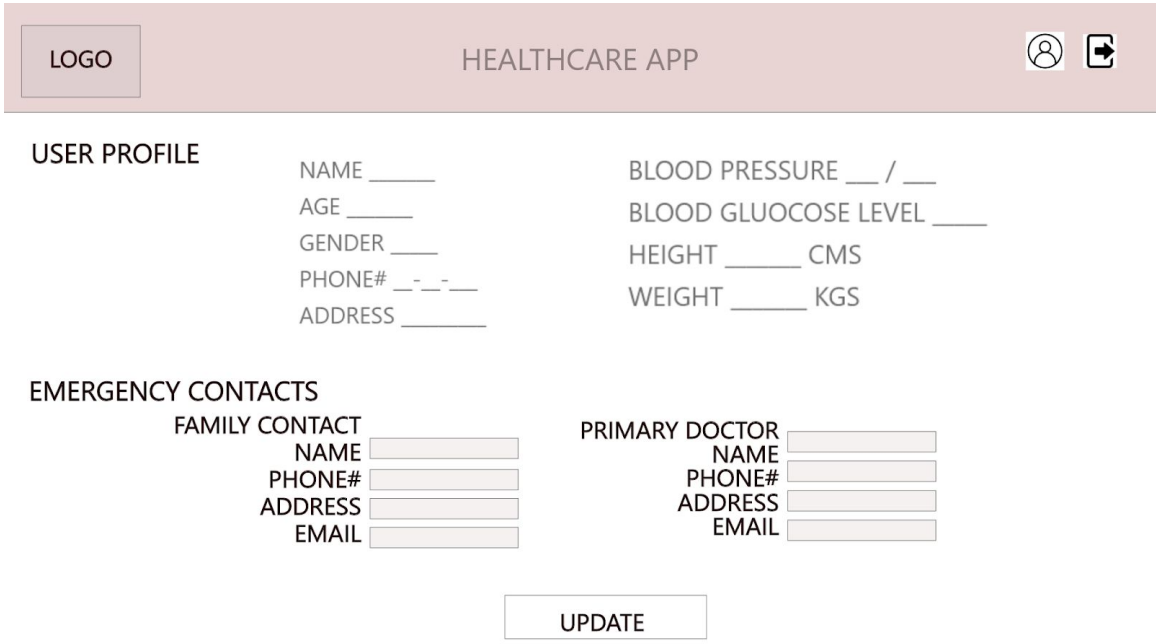


Figure 14: User profile page

Mobile application

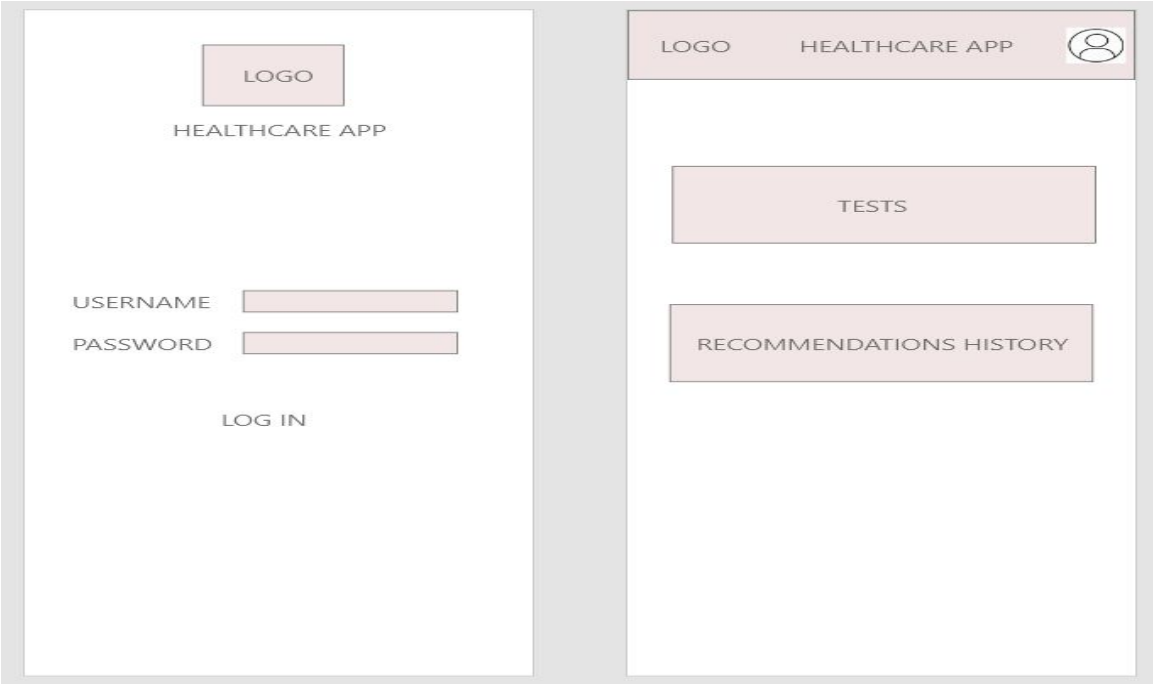


Figure 15: Login page and Main menu page

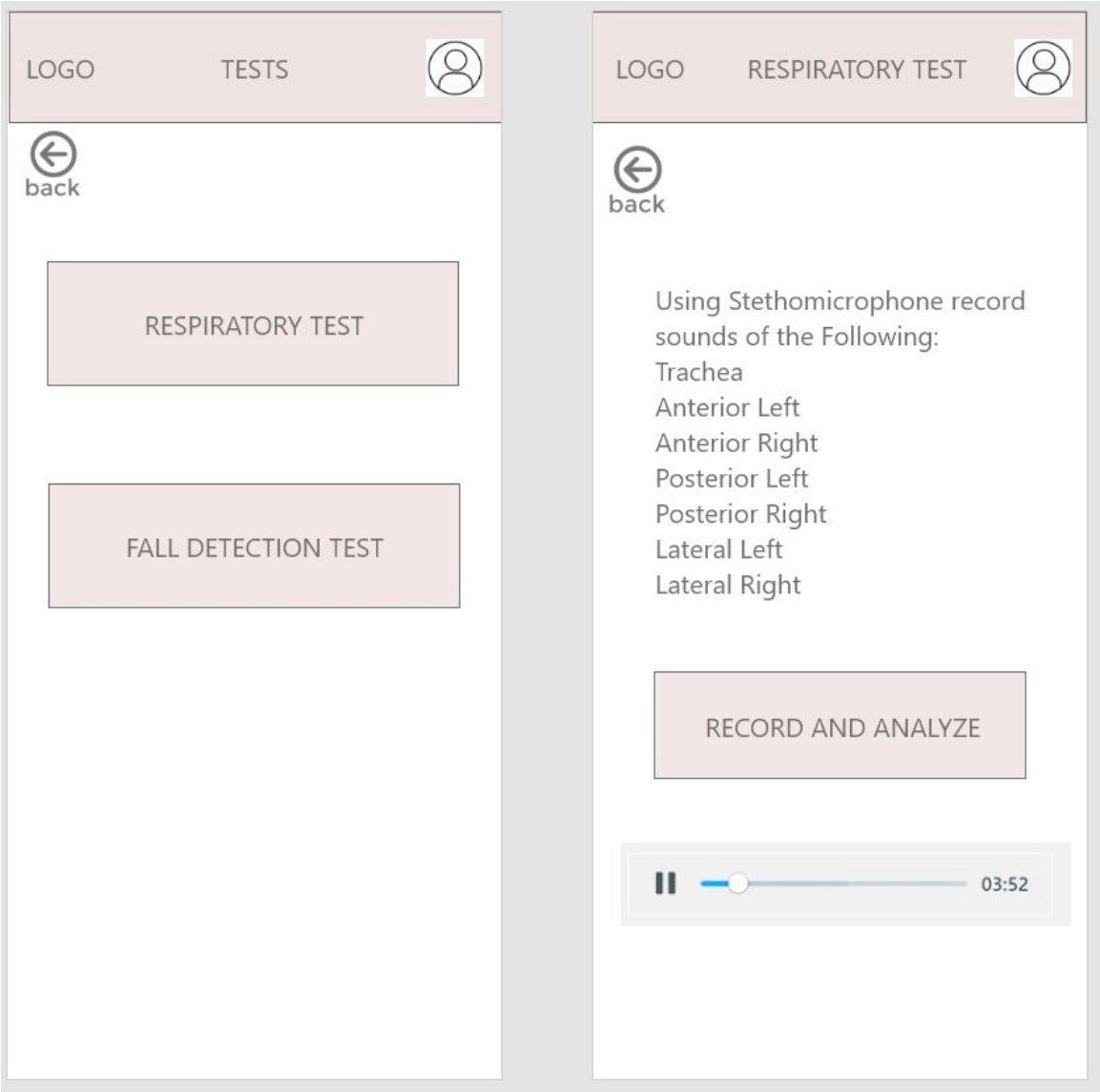


Figure 16: Menu page and respiratory tests page

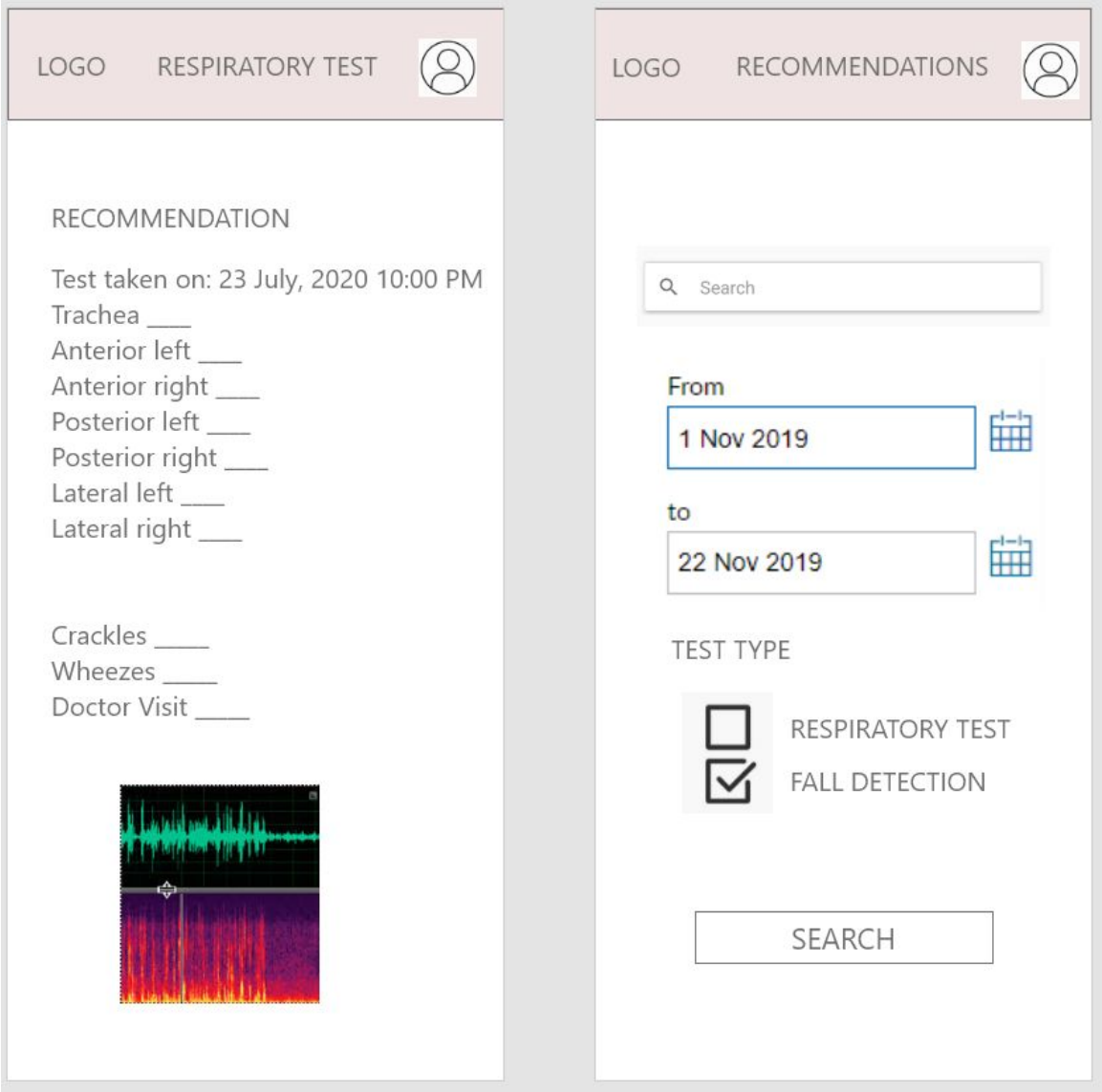


Figure 17: Respiratory recommendation and recommendations search page

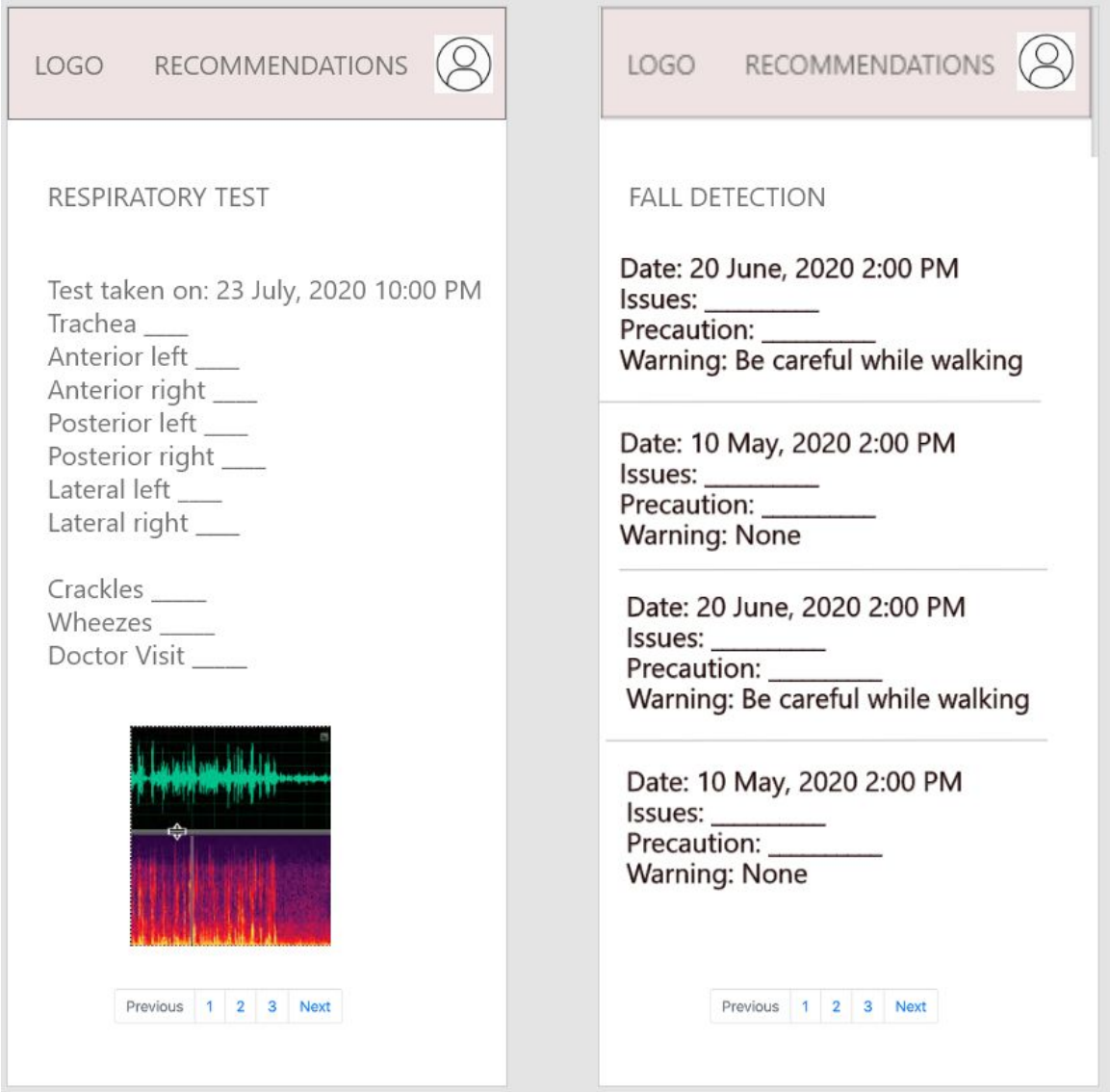


Figure 18: Respiratory tests and fall recommendations page

Figure 19: User profile pages

4.2 Middle-Tier Design

4.2.1 Class Diagram

The figure below depicts the class diagram of the system. The main classes are the User that denotes the users of the app. RespiratoryRecords are the records for respiratory tests and records for fall tests are FallRecords. A user can have multiple respiratory and fall records.

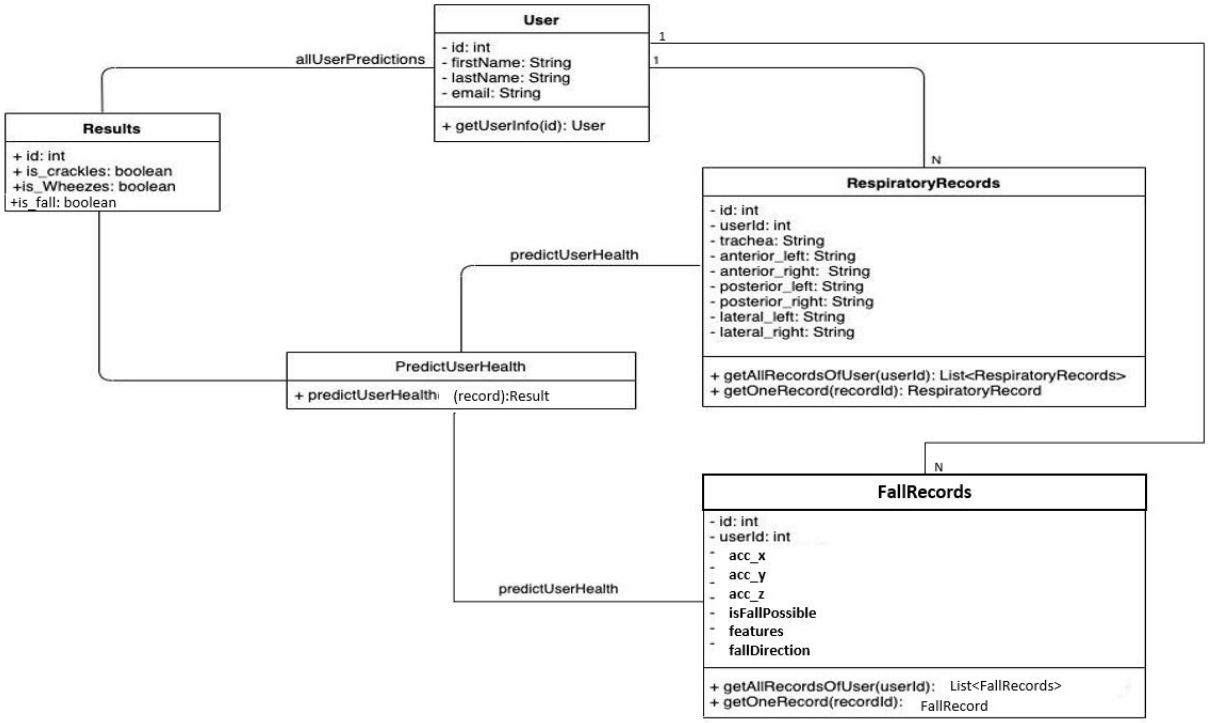


Figure 20: Class diagram for the application

4.2.2 Activity Diagram

Activity Diagram - IoT-Edge Dashboard

The activity diagram of the IoT edge dashboard is depicted below. The interactions between IoT edge and backend systems are shown.

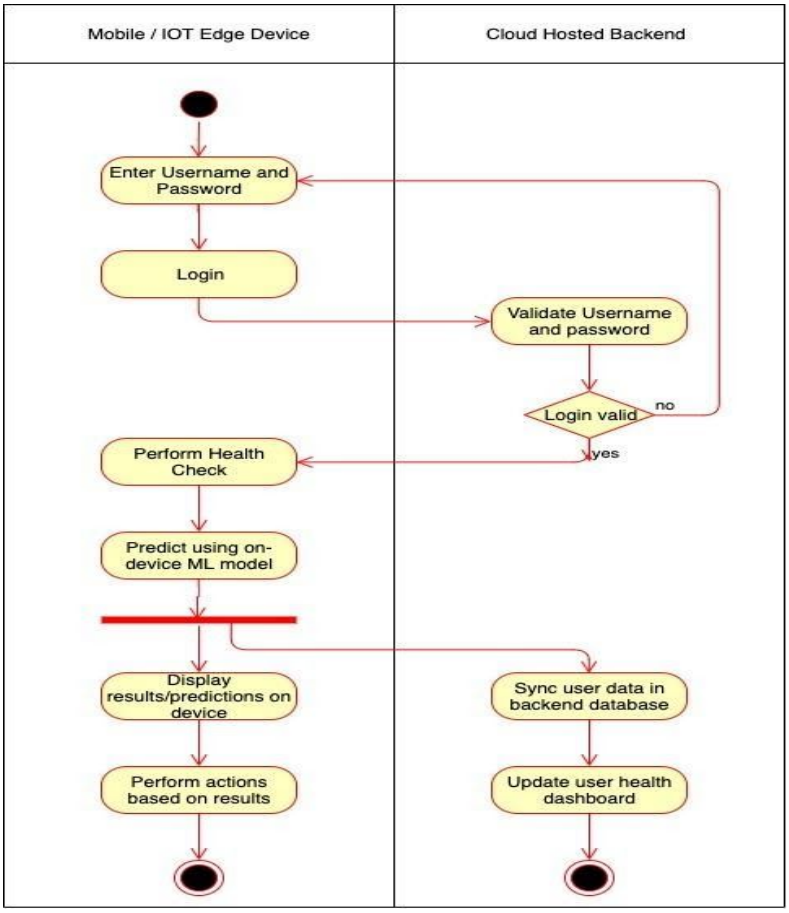


Figure 21: Activity diagram for IoT edge device

Activity Diagram - Web Dashboard

The activity diagram of the web dashboard is depicted below. The interactions between the web dashboard and backend system is shown.

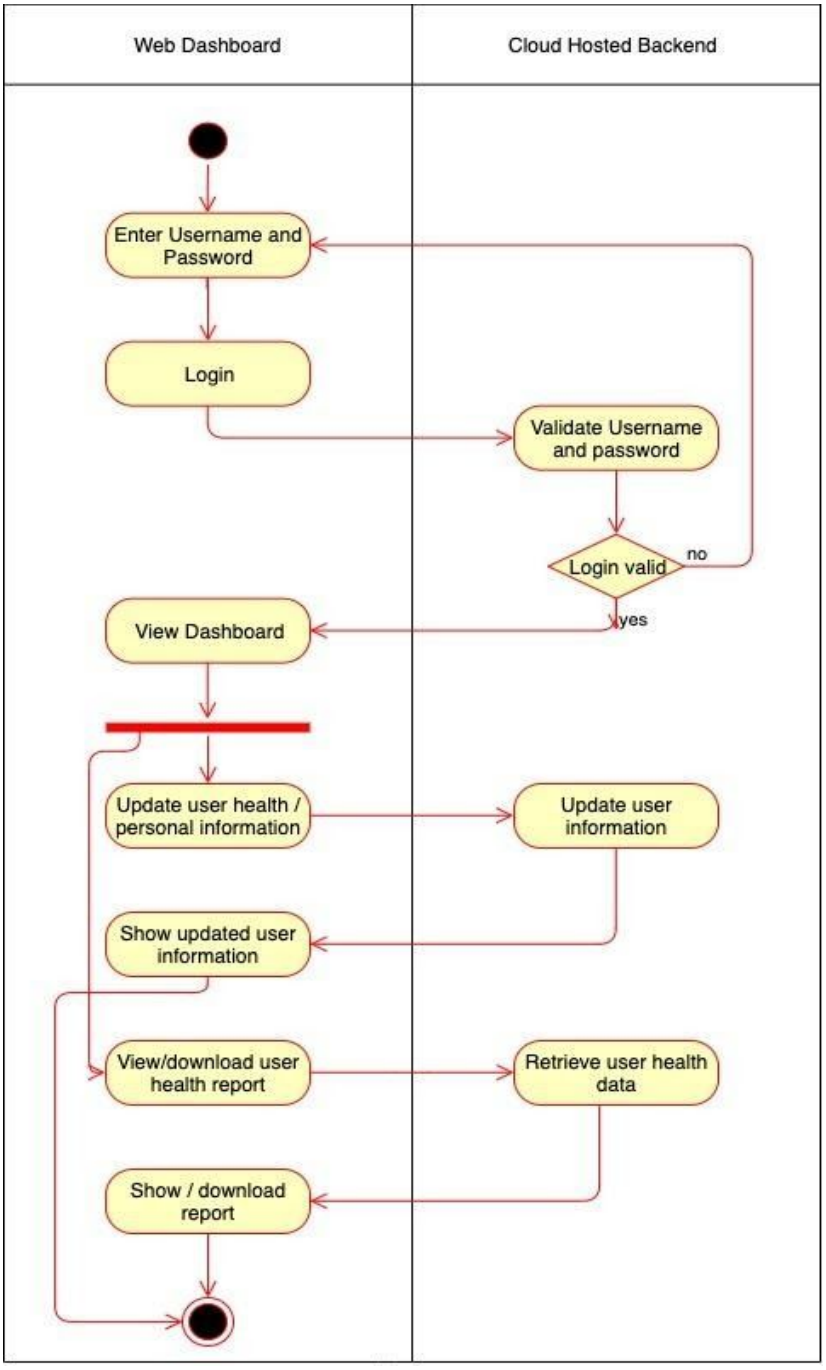


Figure 22: Activity diagram for web dashboard

4.2.3 Hardware Diagram

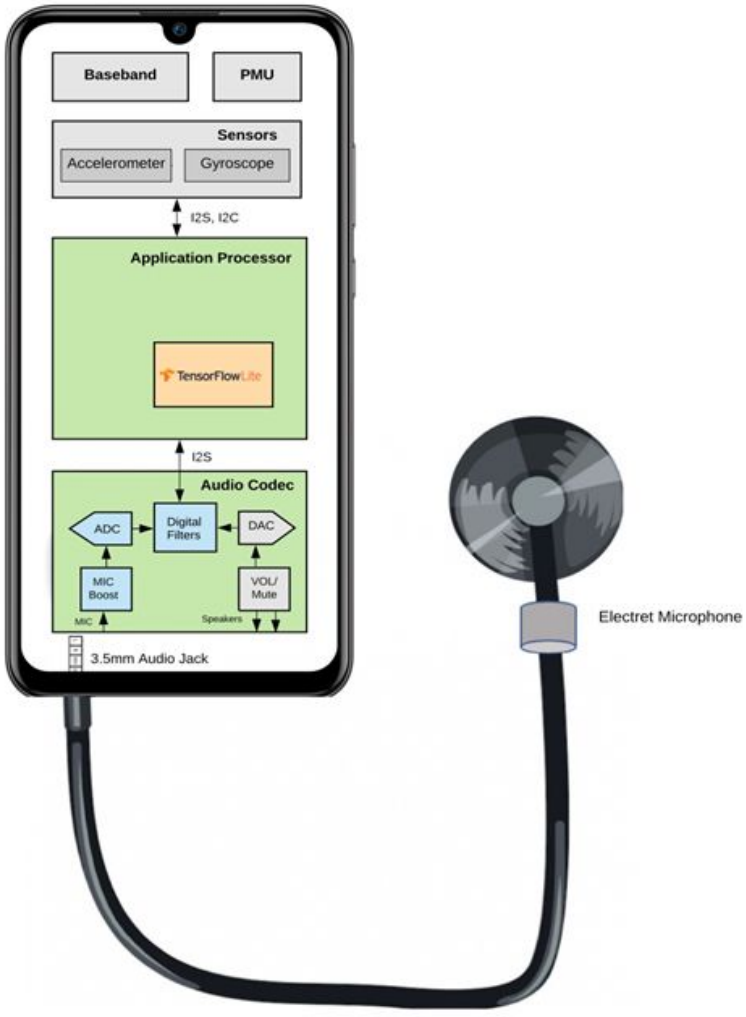


Figure 23: Data acquisition and analysis of lung sounds

For respiratory sound analysis, first, lung sound data is collected using a stethoscope head connected with an electret microphone, something similar to the one in [11]. The microphone converts the audio captured into electrical signals. The microphone is then connected with a 3.5 mm audio cable which will be connected to the mobile phone's audio input. The application framework interacts with audio hardware using android.media APIs. Java Native Interface related to android.media calls native framework which then makes calls to binder IPC. Binder IPC enables communication with an android audio manager which provides audio policy service.

Audio policies interact with audio flinger which has the code that interacts with the audio hardware abstraction layer (HAL). Audio HAL connects the higher-level APIs in android.media with the hardware driver located in the kernel. The signal is then processed in the mobile application, which runs a machine learning model in the background to detect the presence of wheezes or crackles.

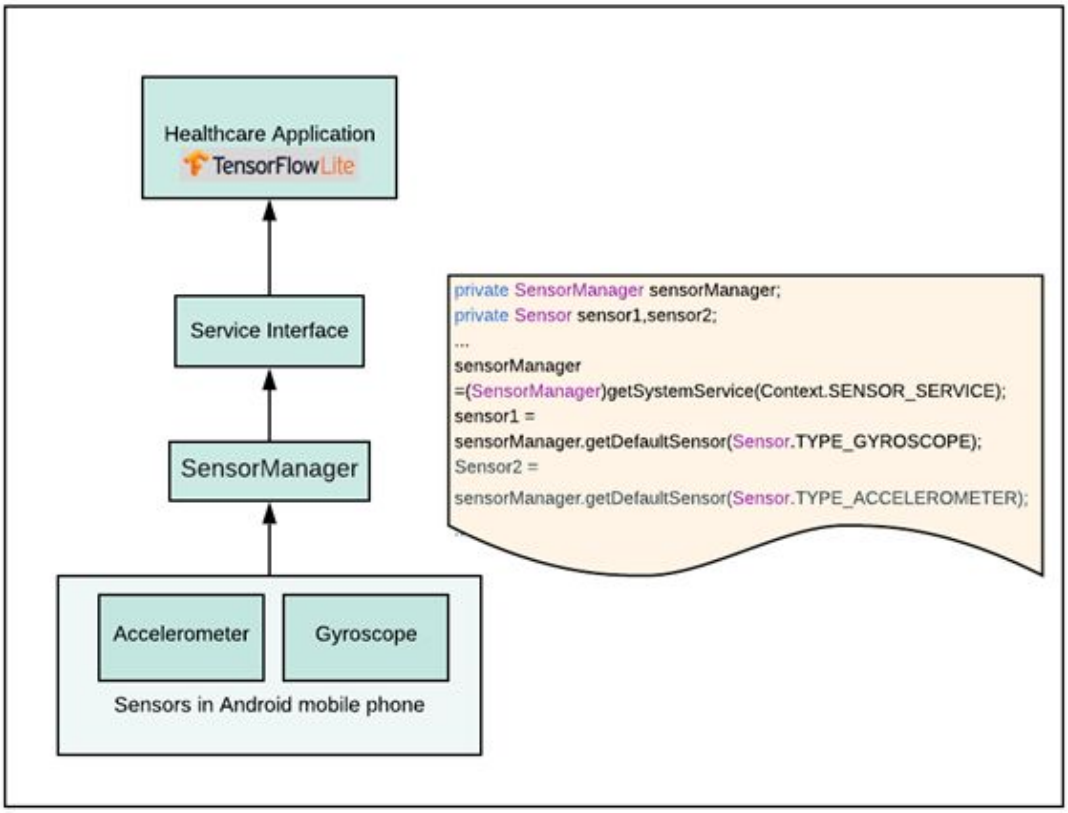


Figure 24: Data acquisition and analysis of falls

For Fall detection and classification, accelerometer and gyro sensors in-built in mobile phones are used. Sensor data are collected by invoking the android sensor service called sensormanager. The collected data is processed in the mobile application. The mobile application runs a machine learning model in the background which detects fall events and provides recommendations.

4.3 Data-Tier Design

This project uses a firebase real-time NoSQL database. For respiratory sound analysis(Figure 25), the database has two collections one is patients, which takes patient-related information as an input, another collection is results, which provides output after the analysis. There are documents related to the patient's information that are contact_info, address, patinet_info, respiratory_records, and emergency_contact. The database design is almost similar for fall detection(Figure 26) except instead of respiratory_records, it has fall_records in its patients' collection. The Moon modeler is used to create a firebase data schema.

4.3.1 Database Schema for Respiratory Sound

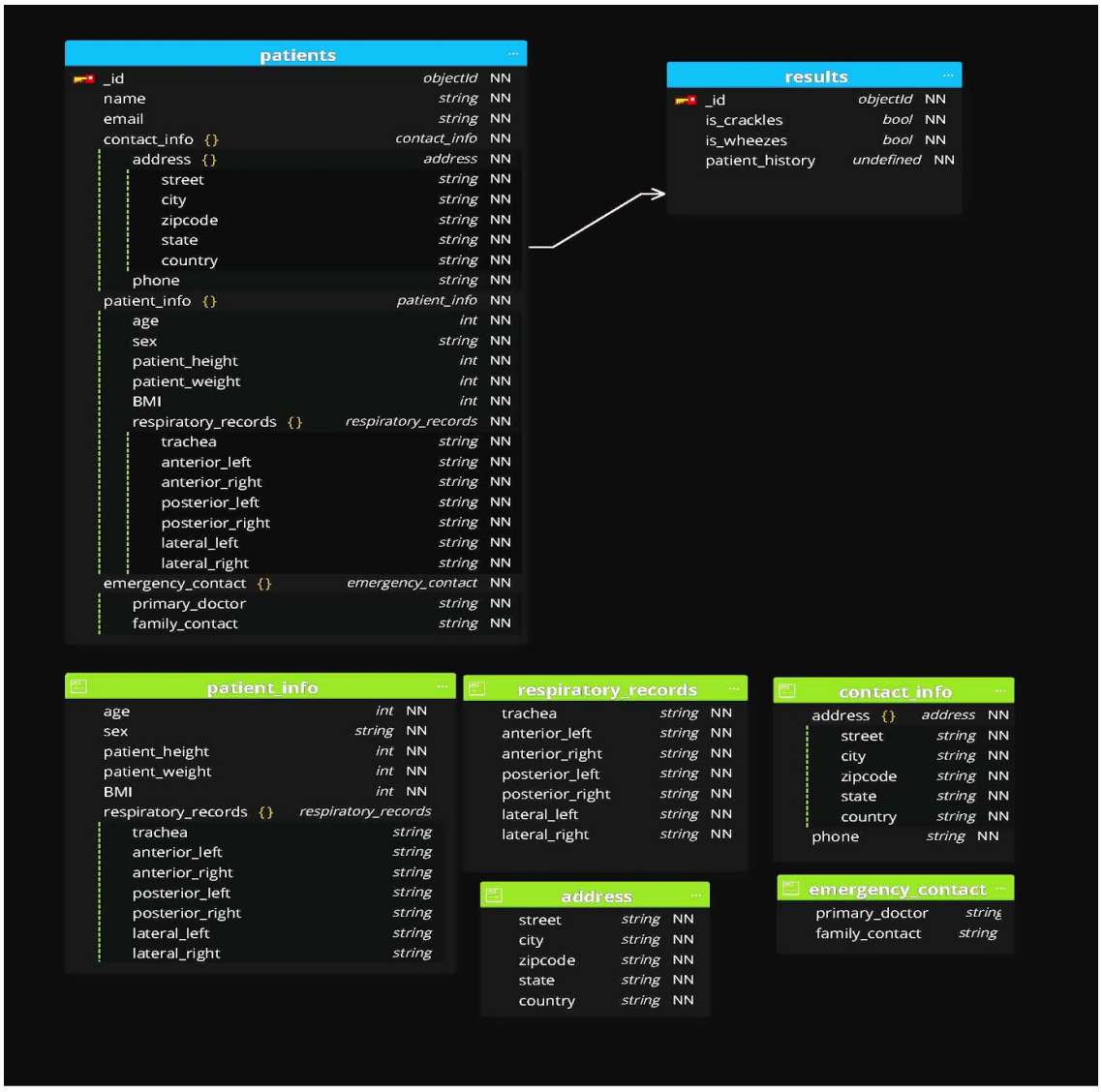


Figure 25: Database schema for respiratory sound

4.3.2 Database Schema for Fall Detection

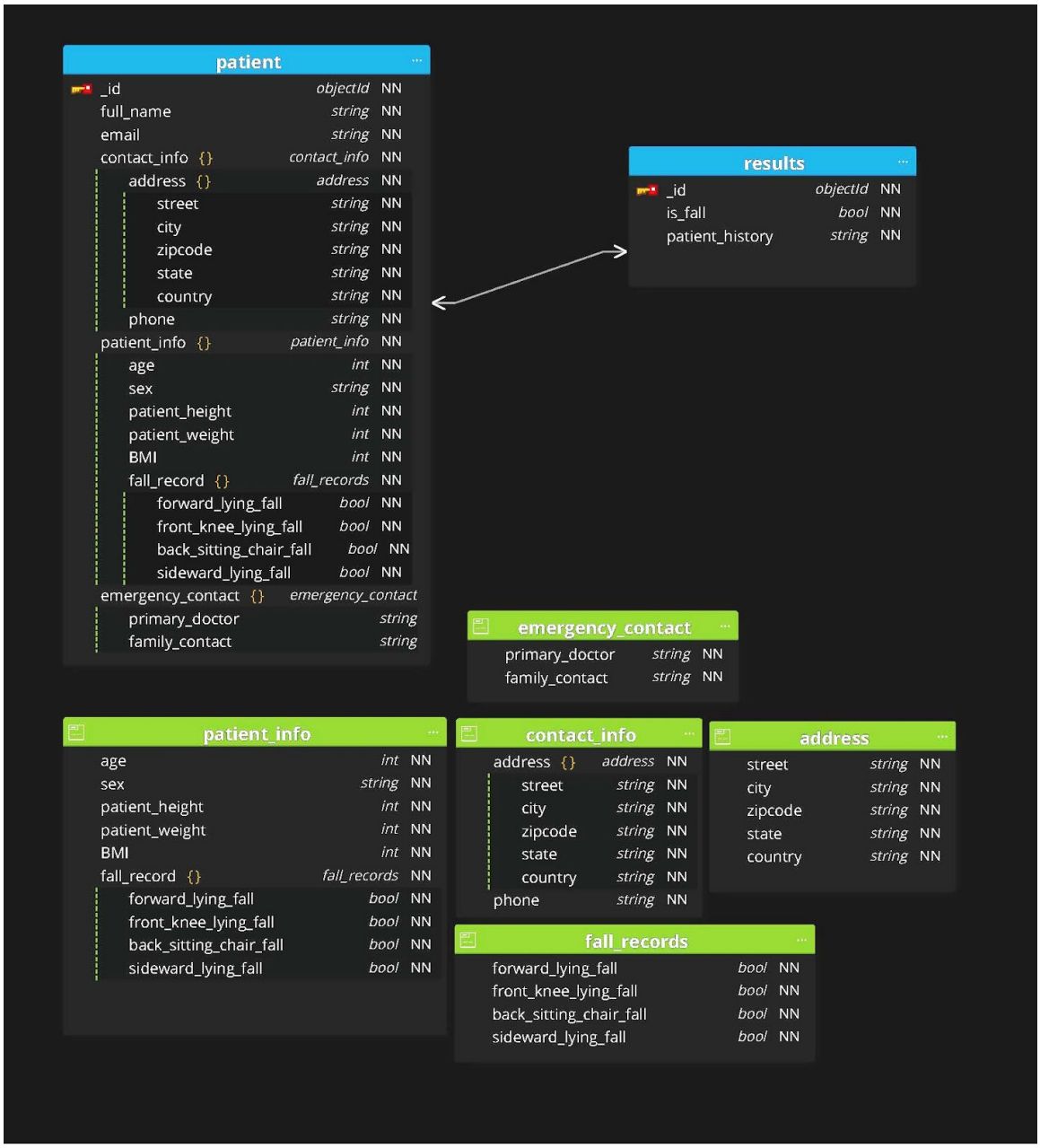


Figure 26: Database schema for fall detection

4.4 Machine Learning Model Design

4.4.1 Preprocessing

The preprocessing flow of the respiratory sound dataset is shown in below figure.

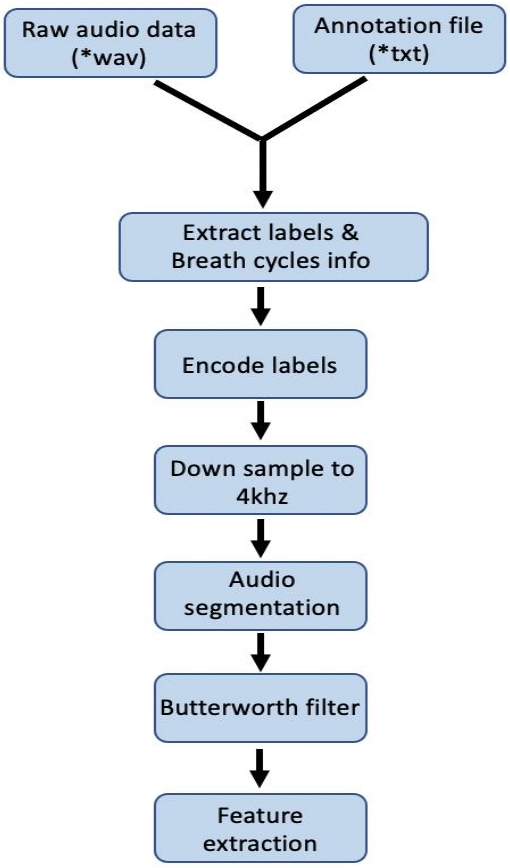


Figure 27: Preprocessing steps - Respiratory dataset

For the respiratory sound dataset, class labels and the starting and ending of respiratory cycles are extracted from the txt files and mapped to the corresponding audio recordings. The labels are encoded into four different categories namely None (0), Crackles (1), Wheeze (2), Both(3). Each audio recording is then down sampled and split into breath cycles. Each breath cycle is passed through a noise removal function and then the processed file is saved as a .wav file for further processing. Fig 28 shows the preprocessing flow of the fall dataset.

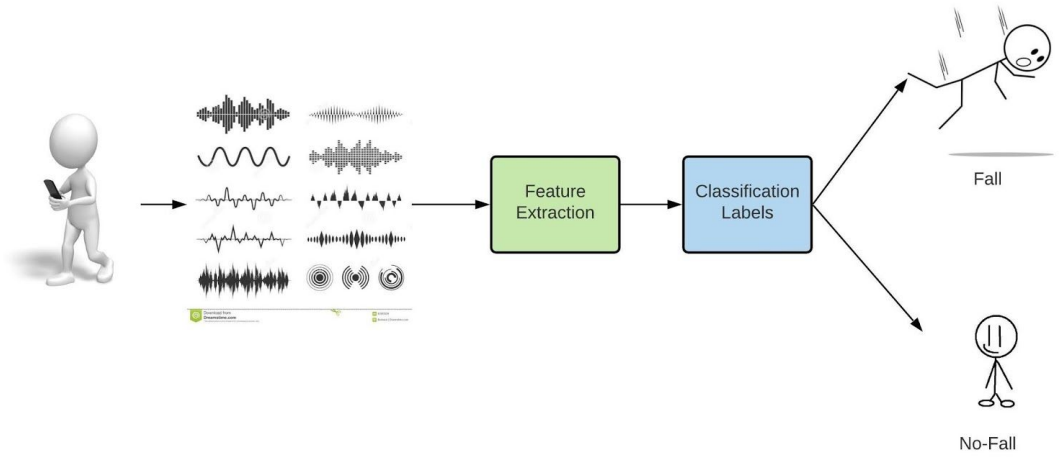


Figure 28: Preprocessing steps- Fall dataset

For the fall dataset, class labels and triaxial angular acceleration magnitude is extracted from the different total acceleration signal provided by the mobile phone. Angular velocity is also measured using a smartphone equipped with a gyroscope. The class labels are encoded into two different categories namely, fall (1) and no-fall (0). Further, the data is converted into csv and used to train a model.

4.4.2 Algorithm Selection and Deployment

Respiratory illness detection

For abnormal lung sound detection, three different neural network architectures namely, 1D CNN, 2D CNN, and LSTM (Long Short-Term Memory) are compared. All three models are trained on the same 80% of the dataset and tested on the remaining 20% of the dataset. The input for the 1D CNN is the mean values of temporal and spectral features of the audio file. Whereas

the 2D CNN and LSTM networks are fed with MFCC features extracted from the audio files. Tuning hyperparameters will result in better accuracy of the model.

Fall event detection

For fall detection, KNN, SVM and Keras Sequential models are compared. The dataset is split into training and testing sets with an 80-20 ratio.

The best algorithm is selected based on the performance accuracy of the test data and converted to tflite format which can be interpreted in mobile applications for making predictions on unseen data.

Chapter 5. Project Implementation

5.1 Client Implementation

5.1.1 Programming and Execution Environment

This project includes two clients, an Android application and a Web application. Both the android application and web application use different programming environments and execution environments. For the android application, the programming language used is java, and the runtime used is ART which stands for Android Runtime. The ART is responsible for translating applications bytecode to native instructions. The development of the web application is done using javascript programming language and the execution environment used by it is Node.js and NPM which stands for Node Package Manager.

	Android Application	**Web Application**
Programming Environment	Java	Javascript
Execution Environment	ART	Node.js and NPM

5.1.2 Web Application

The programming language used to build the web application is Javascript and the framework used is ReactJs. ReactJs is a library to build front-end applications. ReactJs is an open-source library built by engineers at Facebook. React is primarily used to build SPA which stands for single-page applications. Using react we manipulate the DOM of the web browser.

React web application is used as a dashboard for the application, a single place for the customer to find all the statistics and information related to their health which was observed using the mobile phone.

The IDE used to build the web application was a visual studio code, a product of Microsoft. The web application requires authentication and authorization to manage access and to manage the roles of users. The android application uses firebase for authentication and authorization.

Once the application is built it will be deployed to the cloud. The cloud platform used for deploying the web application is Heroku. Also, GIT will be used for version controlling the project.

The dashboard is a read-only dashboard that is used by the user to view the summary of his medical tests and their recommendations. In the dashboard, data synchronized to the Cloud Firestore from the mobile app is used for the dashboard. Firestore library is used to connect to the Firestore database. The collection in the Firestore that is used to store fall tests is fall. The code snippet for connecting to the fall collection in Firestore is shown in the figure.

Connecting to Fall collection

```
async componentWillMount() {
  const fall_collection = db.collection("fall")
  //Get data from Fall Documents
  var fall_data = await fall_collection.where("email", "==", firebase.auth().currentUser.email).get();
  const data = fall_data.docs.map(doc => doc.data());
  data.sort(function (x, y) {
    return x.time.seconds - y.time.seconds;
  })
  var count = 0;

  for (var i = 0; i < data.length; i++) {
    if (data[i].result == "FALL") {
      count++;
    }
  }
```

Figure 29: Code snippet for connecting to fall collection in Firestore

The collection in the Firestore that is used to store respiratory tests and their results is respiratory. The code snippet for connecting to respiratory collection in Firestore is shown in the figure.

Connecting to respiratory database

```
const respiratory_collection = db.collection("respiratory")
var respiratory_data = await respiratory_collection.where("email", "==", firebase.auth().currentUser.email)
const resp_data = respiratory_data.docs.map(doc => doc.data());
resp_data.sort(function (x, y) {
  return x.time.seconds - y.time.seconds;
  })
```

Figure 30: Code snippet for connecting to respiratory collection in Firestore

In the dashboard, summary report, a summary of tests taken and its results is shown. This is a real-time report of the data. Details like the count of tests taken for respiratory and fall is mentioned along with the time of the last test taken. Two pie charts display the results of the two tests taken respectively.

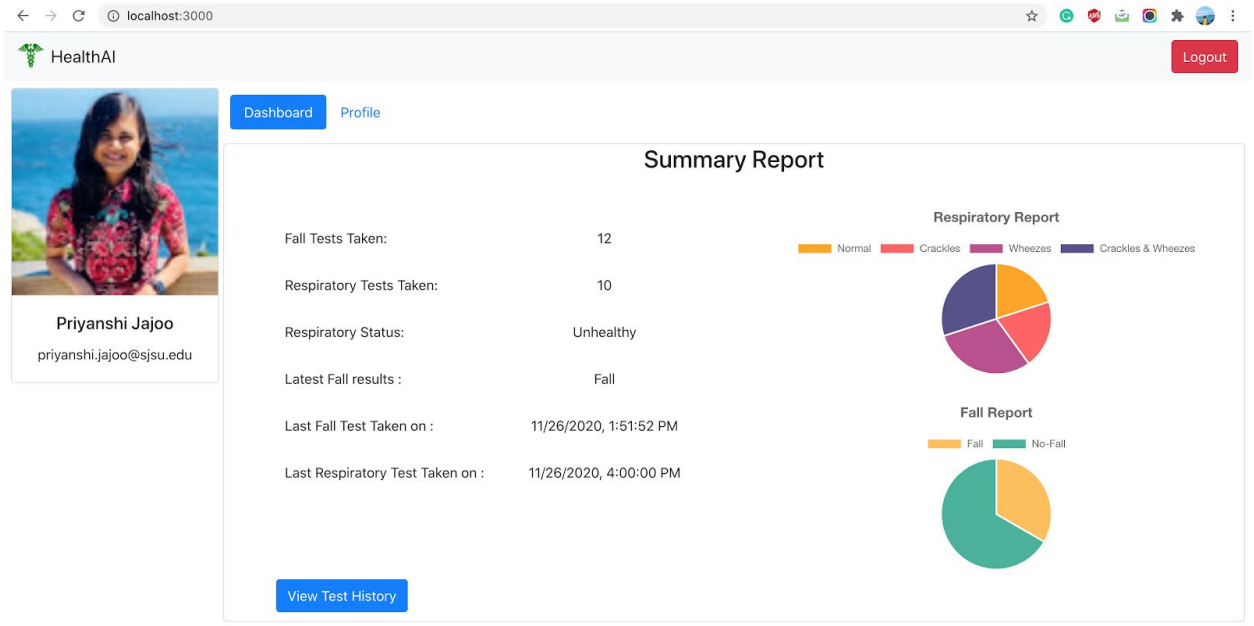


Figure 31: Web dashboard summary report

The pie chart shows the result of the tests taken. It is built using Chart.js, a javascript library, using data from Firestore. This is a real-time graph. For respiratory the various labels used in the chart are Normal if there are no crackles or wheezes detected in the test, Crackles if crackles were detected, Wheezes if wheezes were detected, and Crackles & Wheezes if both were detected in the test.

```
const stateResp = {                    Pie chart component for respiratory activity
  labels: ['Normal', 'Crackles', 'Wheezes', 'Crackles & Wheezes'],
  datasets: [
    {
      label: 'Respiratory Test Report',
      backgroundColor: [
        '#FFA600',
        '#FF6361',
        '#BC5090',
        '#58508D'
      ],
      hoverBackgroundColor: [
        '#2E8B57',
        '#FF69B4',
        '#E97451',
        '#8B0000'
      ],
      data: [countNormal, countCrackles, countWheezes, countBoth]
    }
  ]
}
```

Figure 32: Code snippet for respiratory pie chart

The below figure shows the code snippet for creating the pie chart using Chart.js for fall tests and results data from Firestore. The labels are fall and no fall.

```
//Pie chart values for Fall
const stateFall = {
  labels: ['Fall', 'No-Fall'],
  datasets: [
    {
      label: 'Fall Test Report',
      backgroundColor: [
        '#FFC154',
        '#47B39C'

      ],
      hoverBackgroundColor: [
        '#FFE5B4',
        '#C1E1C1'

      ],
      data: [count, data.length - count]
    }
  ]
}
```

Pie chart component for Fall activity

Figure 33: Code snippet for fall pie chart

When clicked on the View Test History button in the summary report, test history for falls and respiratory tests is shown. Two tables are shown one each for falls and respiratory tests. It lists all the tests taken and shows the timestamp and result for each test.

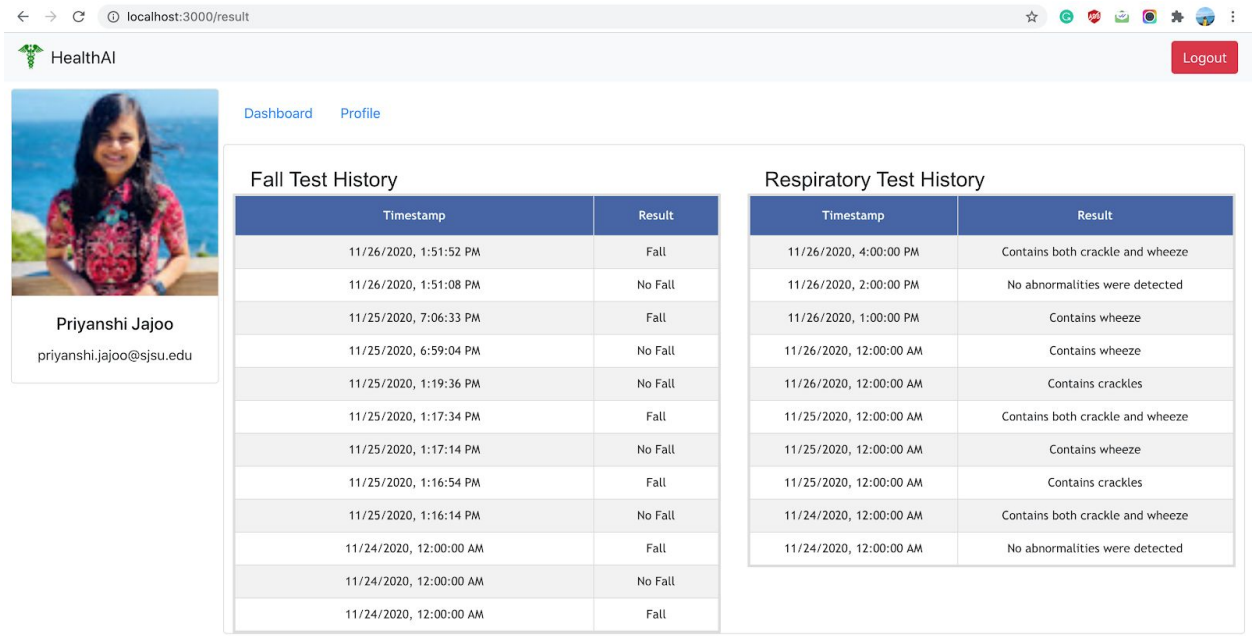


Figure 34: Test history in web dashboard

Using the data from Firestore falls collection, the test history is populated. Each test record is mapped to a row in the table. The below figure shows the code snippet for creating the table using data from the falls collection.

Respiratory tests table

```
renderRespTableData() {
    return this.state.respData.map((respData, index) => {
        const { email, result, time} = respData //destructuring
        var d = new Date(time.seconds * 1000);
        return (
            <tr key={email}>
              <td>{d.toLocaleString()}</td>
                <td>{result}</td>

            </tr>
        )
    })
}
```

Figure 35: Code snippet for respiratory data rendering

Using the data from Firestore respiratory collection, the test history is populated. Each test record is mapped to a row in the table. The below figure shows the code snippet for creating the table using data from the respiratory collection.

Fall test table

```
renderTableData() {
    return this.state.fallData.map((fallData, index) => {
        const { email, result, time} = fallData
        var d = new Date(time.seconds * 1000);
        var fallResult = "";
        if(result=="FALL" || result == "1"){
            fallResult = "Fall";
        }else{
         fallResult = "No Fall";
        }
        return (
            <tr key={email}>
              <td>{d.toLocaleString()}</td>
              <td>{fallResult}</td>

            </tr>
        )
    })
}
```

Figure 36: Code snippet for fall data rendering

In the Profile tab of the dashboard, the user's profile information is displayed in a read-only manner. The details displayed on the page include name, age, height, weight, primary contact, and address. There is also a Logout option for the user to logout of the application.

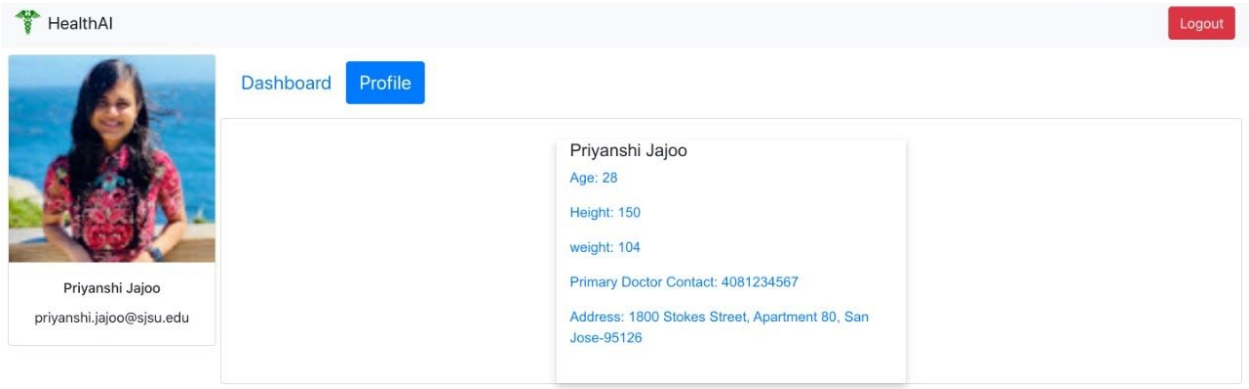


Figure 37: Profile tab in the web dashboard

5.1.3 Mobile Application

The programming language used to build the mobile application is Java and the application uses the android development framework. Android is an open-source mobile framework to build native mobile applications. This framework is developed and maintained by Google.

Using the android application the users will be able to perform two operations. Firstly, they will be able to run a respiratory check. The Respiratory check uses Chaquopy for audio preprocessing and an on-device ML model to predict. Secondly, the user will be able to perform fall detection using the tflite model trained for fall data.

Both the operations fall detection and respiratory check saves the data after operating the database hosted in the cloud. The database used in our case is Firebase's Cloud Firestore, a No-SQL cloud-hosted database by Google.

After the prediction by the machine learning models, an SMS notification is sent to the user's contact regarding the test status. It is implemented using SmsManager and the SMS is sent from the user's phone number to the contact.

The final application which is ready to release into production will be converted to an APK that is ready to be published to the Android Play Store.

App Constraints

- ***Software constraints***

  For respiratory illness classification, Python's Scipy and librosa packages were used for preprocessing and audio feature extraction. Equivalent signal processing and feature extraction libraries were not available in java. So, for implementing the mobile application, Chaquopy (Python SDK for Android) was used. Chaquopy offers open-source licenses for free of charge upon releasing the app source code on github.

- ***Hardware constraints***

  A stethoscope with audio jack as shown in [11] is required to record lung sounds in the mobile phone. Due to the unavailability of the product, lung sound samples reserved from the dataset are used for mobile application testing.

Android environment setup

Android Studio is the official Integrated Development Environment (IDE) for Android app development launched by Google, based on IntelliJ IDEA. As part of the Android environment setup downloaded and installed the following software

- Java Development Kit
- Android SDK
- Java runtime environment

Activities

Configured emulator using the Android Virtual Device (AVD) manager and created a new android studio project. Three different activities were created.

- MainActivity.java
- RespiratoryActivity.java
- FallActivity.java

MainActivity.java contains the code for user Sign in and two buttons namely, “PERFORM RESPIRATORY CHECK” and “DETECT FALL”.

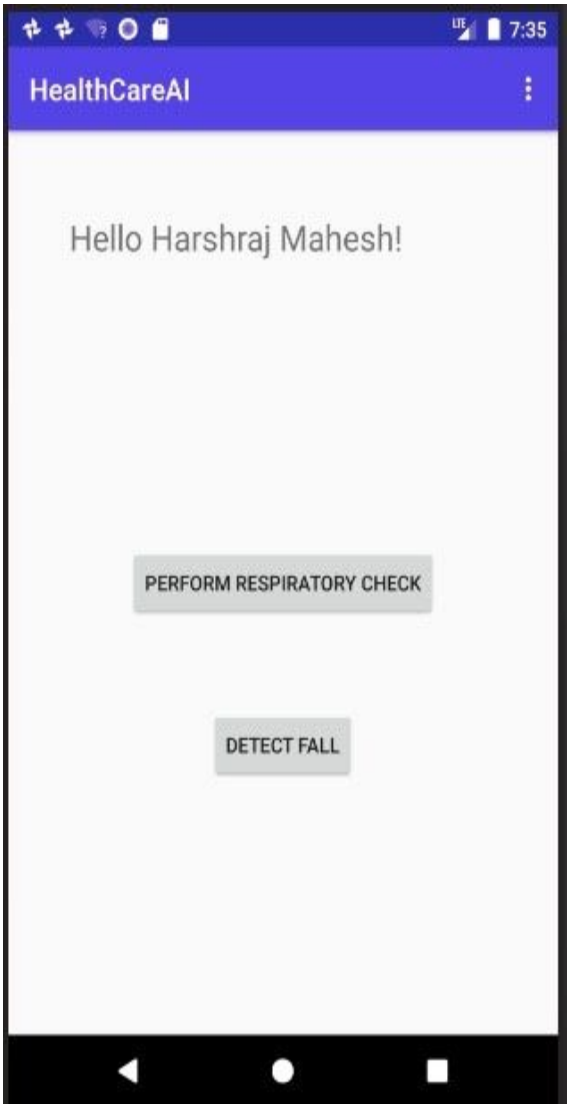


Figure 38: MainActivity user interface

Respiratory Test

Clicking “PERFORM RESPIRATORY CHECK” calls the RespiratoryActivity which has the “SELECT AUDIO” button. On clicking the “SELECT AUDIO” button the user is directed to device storage from which the user can navigate and select an audio for processing. This is accomplished by using Intent() and URI objects.

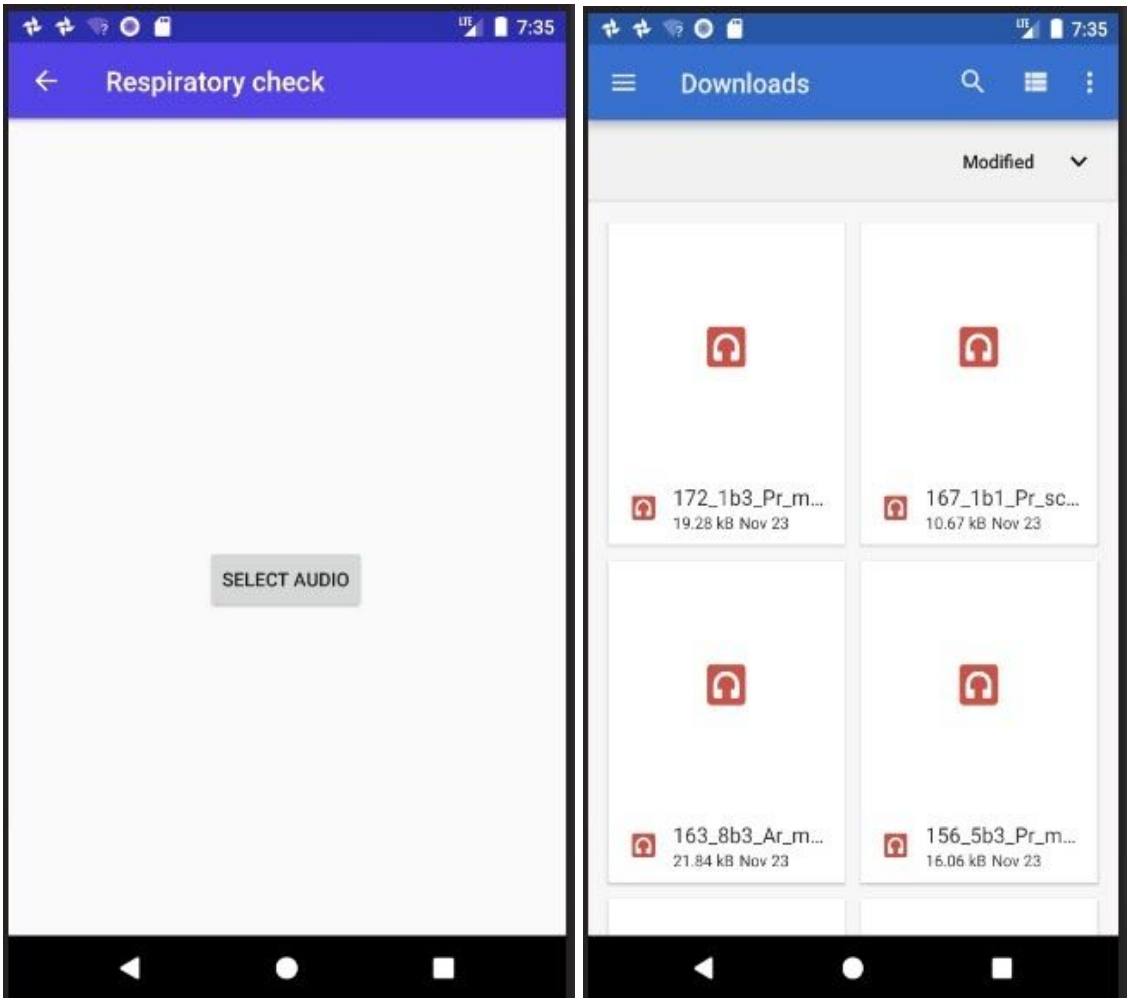


Figure 39: RespiratoryActivity and file chooser UI

After the file is selected, the file path is extracted and passed to the featureExtraction.py using Chaquopy's Python Object.

```
if (py == null){
    py = Python.getInstance();
    pyobj = py.getModule( s: "featureExtraction");
}
String src = urls[0].getPath();
String[] path = src.split( regex: ":");
PyObject obj = pyobj.callAttr( key: "build_feat",path[1]);
```

Figure 40: Creating Chaqoupy instance and calling preprocessing file

The featureExtraction.py file performs the preprocessing steps such as converting the audio to mono channel, downsampling to 4000 Hz, and extracting the MFCC features. The extracted features stored in a Python object are converted to java class and stored in a java array. The java array is passed as an input to tflite interpreter using tflite.run. The predictions results are shown on screen.

Figure 41: Prediction results

Further a SMS notification is sent to the primary contact or the caregiver upon predicting alarming results.

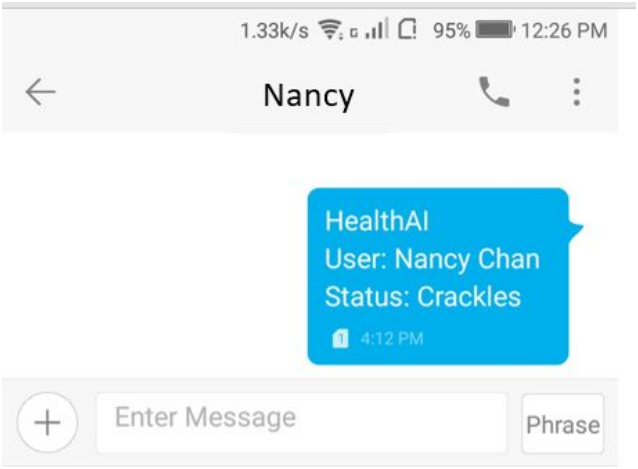


Figure 42: SMS notification received

Fall Test

Clicking “DETECT FALLS” button in MainActivity calls the FallActivity which starts to record sensor values using sensor manager for the given number of instances. The collected sensor values are then preprocessed and passed as an input to the tflite model. The prediction results are displayed on screen and an SMS notification is sent to the caregiver if fall is detected.

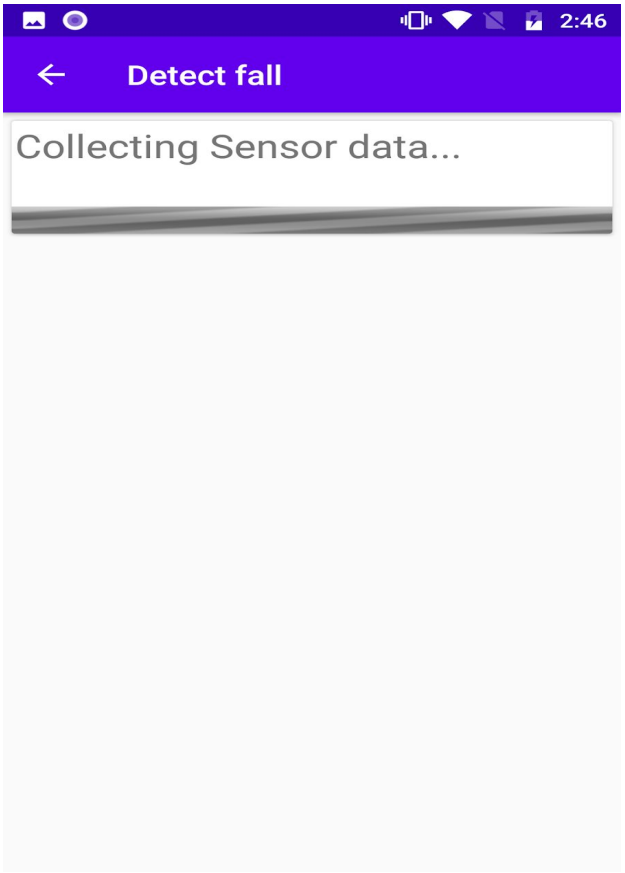


Figure 43: Fall activity and collecting sensor data

Sensor data has been collected for 100 instances. Figure 44 and 45, shows the mathematical calculations for the fall prediction model.

```
FirebaseFirestore db = FirebaseFirestore.getInstance();
SensorManager mSensorManager;
Sensor mAccelerometer;
Sensor mGyroscope;
int sensorDataCount =100;
String phoneNumber = "4087149213";
String[] sensorData = new String[sensorDataCount];
```

Figure 44: Fall activity sensor data collection

```
float[][] output = new float[1][1];
tflite.run(inputFeatures, output);
float inferredValue = output[0][0];
Log.d( tag: "Value: ", String.valueOf(inferredValue));
int result = Math.round(inferredValue);
```

Figure 45: Fall prediction

Prediction results are shown on the screen as shown in the figure 46.

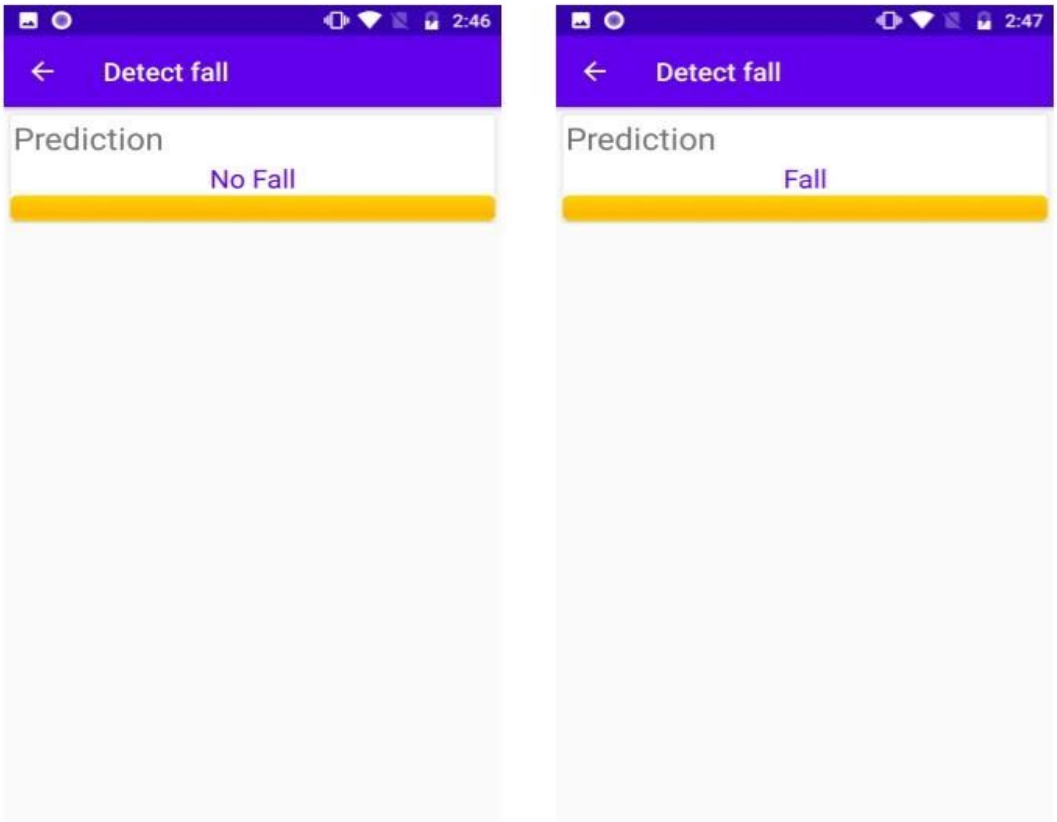


Figure 46: Prediction results

5.2 Data-Tier Implementation

5.2.1 Database Operations for Client

The clients (Android application and Web Application) perform Database related operations to save customer data. The saved customer data is used to build a Web dashboard that can be used by the user to track progress, get a detailed view, and to visualize the user's health-related statistics.

The database operations performed by the Android application client are as follows:

- Create an entry after respiratory check
- Create an entry after Fall detected
- Update user information

The database operations performed by the Web application client are as follows:

- Fetch respiratory check entries of a given user
- Fetch fall detection entries of a given user
- Update user information

Both the Web and Android application uses Firebase API to store and retrieve data in NoSQL format.

5.3 Machine Learning Model Implementation

This section explains the end to end implementation of machine learning models right from the data collection until trained tflite model creation.

5.3.1 Respiratory Illness Detection

Data collection

ICBHI dataset is used for respiratory sound analysis [12] which was organized to aid the scientific contest at Int. Conf. on Biomedical Health Informatics. The dataset contains audio samples acquired by two independent research teams from two different countries. The dataset consists of respiratory sounds collected from different equipment, various chest locations, and noise artifacts which replicates real-time conditions. The dataset contains audio recordings as .wav files and the corresponding annotations in .txt file which holds the start and end time of breath cycles and the presence of crackles and wheezes. The recordings were annotated manually by respiratory-related health professionals.

Dataset also includes patient diagnosis and demographic information. Demographic info has the following details,

- Patient number
- Age
- Sex
- Adult BMI (kg/m2)
- Child weight (kg)
- Child Height (cm)

Each audio file has been named using a specific naming convention explained in the table below,

#	Convention	Comments
1.	Patient number	Unique for all the patients
2.	Recording index	Taken from the audio recordings
3.	Chest location	1. Trachea

		2. Anterior left (Al) 3. Anterior right (Ar) 4. Posterior left (Pl) 5. Posterior right (Pr) 6. Lateral left (Ll) 7. Lateral right (Lr)
4.	Acquisition mode	1. sequential/single channel (sc) 2. simultaneous/multichannel (mc)
5.	Recording equipment	1. AKG C417L Microphone (AKGC417L) 2. 3M Littmann Classic II SE Stethoscope (LittC2SE) 3. 3M Littman 3200 Electronic Stethoscope (Litt3200) 4. WelchAllyn Meditron Master Elite Electronic Stethoscope (Meditron)

Table 1: Respiratory dataset summary

Exploratory Data Analysis

The statistics summary derived from the dataset provides better insights which will help to choose the right preprocessing steps. The dataset includes 920 recordings of varying length ranging from 10s to 90s. These recordings were taken from 126 patients.

Figure 47 shows the no.of recordings acquired from each equipment. Most of the recordings are acquired using AKGC417L.

```
Device : Recordings
AKGC417L    646
Meditron    127
LittC2SE     87
Litt3200     60
Name: device, dtype: int64
```

Figure 47: Device vs number of recordings

Figure 48 shows the no.of recordings in each frame rate. Most of the recordings are 44.1KHz, yet for the detection of crackles and wheezes sampling rate of 4KHz is sufficient.

```
Frates : Recordings
44100    824
4000      90
10000      6
Name: frate, dtype: int64
```

Figure 48: Framerate vs the number of recordings

Figure 49 shows the no.of recordings in each sample width available

```
SampleWidth (Bytes) : Counts
3    792
2    128
Name: sampleWidth, dtype: int64
```

Figure 49: Sample width vs number of recordings

Figure 50 shows the breath cycle count of each class label.

Class label	Count
Normal	3642
Crackles	1864
Wheezes	886
Both	506

Figure 50: Cycle count

The below figure shows the histogram of the duration of breath cycles. As seen earlier, the average duration of the breath cycle is 2.7s.

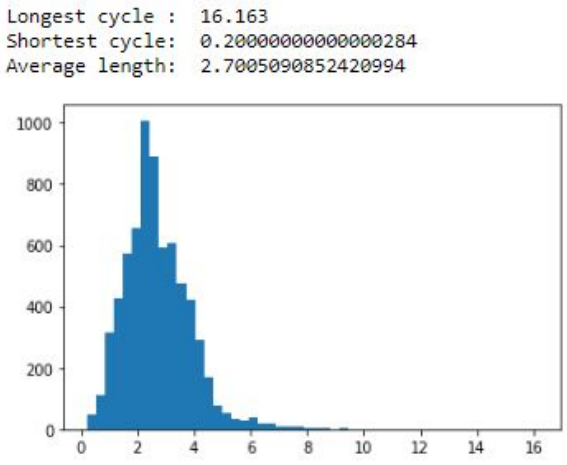


Figure 51: Cycle duration

It is clear from the above analysis that the data is highly imbalanced and collected from heterogeneous sources which make the classification task more challenging

Data Visualization

Waveplot (amplitude vs time), power spectrum (magnitude Vs frequency), and spectrogram (frequency vs time) graphs are shown for sample audio from each of the class labels. As seen in the figure, “wheeze” and “both” (Crackles and wheeze) classes are hard to differentiate because of the noise artifacts.

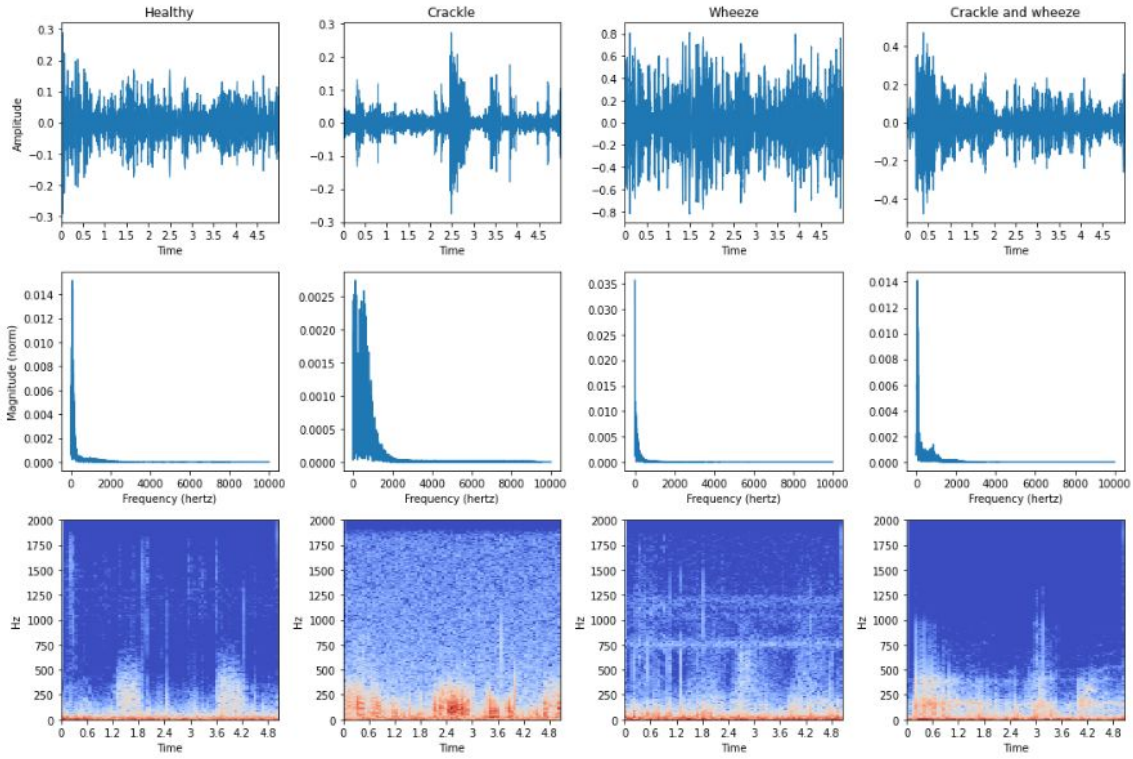


Figure 52: Waveplot, power spectrum, and spectrogram

Data Preprocessing

Data Cleaning

In this step, the raw audio data is processed and transformed into NumPy arrays which can be used for training the neural networks.

As the first step of preprocessing the audio recordings are mapped with the annotation file and a Pandas dataframe is created.

	fname	start	end	crackles	wheezes
0	101_1b1_Al_sc_Meditron	0.036	0.579	0	0
1	101_1b1_Al_sc_Meditron	0.579	2.450	0	0
2	101_1b1_Al_sc_Meditron	2.450	3.893	0	0
3	101_1b1_Al_sc_Meditron	3.893	5.793	0	0
4	101_1b1_Al_sc_Meditron	5.793	7.521	0	0

Figure 53: audio files and annotated information

In the next step, the crackles and wheezes class is encoded to represent labels in a one-dimensional format.

```
#Encode crackles and wheezes in single column
crack_wheez = []
for idx, row in file_data.iterrows():
    if row['crackles'] == 0 and row['wheezes'] == 0:
        crack_wheez.append(0)
    elif row['crackles'] == 1 and row['wheezes'] == 0:
        crack_wheez.append(1)
    elif row['crackles'] == 0 and row['wheezes'] == 1:
        crack_wheez.append(2)
    else:
        crack_wheez.append(3)
```

file_data

	fname	start	end	crack_wheez
0	101_1b1_Al_sc_Meditron	0.036	0.579	0
1	101_1b1_Al_sc_Meditron	0.579	2.450	0
2	101_1b1_Al_sc_Meditron	2.450	3.893	0
3	101_1b1_Al_sc_Meditron	3.893	5.793	0
4	101_1b1_Al_sc_Meditron	5.793	7.521	0
...	...	...	...	...
6893	226_1b1_Pl_sc_LittC2SE	11.721	13.693	1
6894	226_1b1_Pl_sc_LittC2SE	13.693	15.536	0
6895	226_1b1_Pl_sc_LittC2SE	15.536	17.493	0
6896	226_1b1_Pl_sc_LittC2SE	17.493	19.436	1
6897	226_1b1_Pl_sc_LittC2SE	19.436	19.979	0

6898 rows × 4 columns

Figure 54: Encoded class labels 0 (None), 1 (Crackle), 2 (Wheeze), 3 (Both)

Audio recordings are downsampled to 4000Hz such that the signal ranges from 0 to 2000Hz which is the required range for detection of crackles and wheezes.

To remove the environmental noise such as equipment's contact noise and other interferences, a 12th order Butterworth bandpass filter with cutoff frequencies 100Hz to 1800 Hz is chosen as it achieved the best results in the official paper published by the authors of this dataset. The filter is implemented using Python's Scipy module.

Each audio recording is passed through the pipeline where it is loaded, downsampled, split into breath cycles, and filtered using a bandpass filter. The filtered signal is saved as '.wav' format using Python's Soundfile library. The saved '.wav' files can be loaded again for feature extraction.

	fname	start	end	crack_wheez	fname_cycle
0	101_1b1_Al_sc_Meditron	0.036	0.579	0	101_1b1_Al_sc_Meditron_0.wav
1	101_1b1_Al_sc_Meditron	0.579	2.450	0	101_1b1_Al_sc_Meditron_1.wav
2	101_1b1_Al_sc_Meditron	2.450	3.893	0	101_1b1_Al_sc_Meditron_2.wav
3	101_1b1_Al_sc_Meditron	3.893	5.793	0	101_1b1_Al_sc_Meditron_3.wav
4	101_1b1_Al_sc_Meditron	5.793	7.521	0	101_1b1_Al_sc_Meditron_4.wav
...	...	...	...	...	...
6893	226_1b1_Pl_sc_LittC2SE	11.721	13.693	1	226_1b1_Pl_sc_LittC2SE_6.wav
6894	226_1b1_Pl_sc_LittC2SE	13.693	15.536	0	226_1b1_Pl_sc_LittC2SE_7.wav
6895	226_1b1_Pl_sc_LittC2SE	15.536	17.493	0	226_1b1_Pl_sc_LittC2SE_8.wav
6896	226_1b1_Pl_sc_LittC2SE	17.493	19.436	1	226_1b1_Pl_sc_LittC2SE_9.wav
6897	226_1b1_Pl_sc_LittC2SE	19.436	19.979	0	226_1b1_Pl_sc_LittC2SE_10.wav

6898 rows × 5 columns

Figure 55: Pandas dataframe with preprocessed filenames

The dataset is split into training and testing sets using the train test split method from Scikit learn model selection with random state initialization.

Feature extraction

Two different modules are used for feature extraction. The first one is pyAudioAnalysis [14] which provides basic audio functionality and a short-term feature extraction method using which the features related to time and frequency components of the audio files are extracted. Figure 56 shows the description of the extracted features. The mean values of the extracted features are used as input for the 1D convolutional network.

Feature ID	Feature Name	Description
1	Zero Crossing Rate	The rate of sign-changes of the signal during the duration of a particular frame.
2	Energy	The sum of squares of the signal values, normalized by the respective frame length.
3	Entropy of Energy	The entropy of sub-frames' normalized energies. It can be interpreted as a measure of abrupt changes.
4	Spectral Centroid	The center of gravity of the spectrum.
5	Spectral Spread	The second central moment of the spectrum.
6	Spectral Entropy	Entropy of the normalized spectral energies for a set of sub-frames.
7	Spectral Flux	The squared difference between the normalized magnitudes of the spectra of the two successive frames.
8	Spectral Rolloff	The frequency below which 90% of the magnitude distribution of the spectrum is concentrated.
9-21	MFCCs	Mel Frequency Cepstral Coefficients form a cepstral representation where the frequency bands are not linear but distributed according to the mel-scale.
22-33	Chroma Vector	A 12-element representation of the spectral energy where the bins represent the 12 equal-tempered pitch classes of western-type music (semitone spacing).
34	Chroma Deviation	The standard deviation of the 12 chroma coefficients.

Figure 56: List of features extracted from pyAudioAnalysis

For the 2D CNN and LSTM networks, MFCC features [15] are used. MFCC stands for Mel Frequency Cepstral Coefficients which typically involves the following steps,

- windowing the signal
- Apply discrete Fourier transform
- Take a log of the magnitude
- Warp frequencies on a Mel scale
- Apply inverse discrete cosine transform

Typically MFCC contains 40 dynamic coefficients. The 0th coefficient is often omitted because it represents the average log-energy and does not carry significant speaker-specific information. In Mel Frequency Spectrum, the frequency bands are spaced equally on the mel scale which is very close to human perception for audio sensitivity. MFCC features are the commonly used and proven method for audio processing and speech recognition.

Python librosa module offers an MFCC feature extraction method that performs all the above steps and outputs the coefficients. The list of parameters of librosa MFCC feature are given below,

- n_fft=512 (number of FFT components)
- win_length=400 (Window length)
- n_mfcc=20 (Number of DCT coefficients to keep)
- hop_length = 256 (Number of samples to overlap)
- n_mels = 128 (Number of mel filters)
- fmin = 100, fmax = 1800 (Cutoff range for Y-axis spectrum)

Machine learning models

Different machine learning architectures are compared by training on the same 80% of the audio data and testing on the remaining 20%. Though the input to the model varies, the output layer remains the same for all the models. The output layer has 4 (Number of classes) units with softmax activation function which outputs the probability of output belonging to each class. All of the models are compiled with categorical cross-entropy loss and adam optimizer.

1. Depth Separable 1D CNN

In depth wise convolutional architectures, the convolution operation is applied to one channel at a time. This is different from traditional CNN's where the convolution is performed for all the channels at the same time. The main advantage of this technique is the lesser number of

parameters which essentially reduces overfitting. Depth separable networks are computationally inexpensive and a lot faster than standard CNN.

Figure 57 shows the 1D separable convolution model. The input to the model is the mean values of short-term features extracted from the pyAudioAnalysis library. Using the mean values of the extracted feature allows the use of audio recordings of varying duration. There are 34 features hence the input shape is (34, 1).

Tanh activation is used for the first layer as it normalizes the result in the range of (-1,1). For the following layers, the Relu activation function is used which is very common in deep neural networks. The major benefit of this activation is the neurons are activated only if the output of the linear transformation is greater than 0 which makes it computationally efficient.

The Sequential model is further stacked with max-pooling and a dropout layer. After the first dropout layer, a Separable 1D Conv layer is added with 256 filters and a kernel size of 5. This is followed by a max-pooling and dropout layer. Global average pooling is used to reduce the spatial dimension and then follows the fully connected layer with 512 filters.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 30, 64)            384
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 26, 128)           41088
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 13, 128)           0
_________________________________________________________________
dropout_1 (Dropout)          (None, 13, 128)           0
_________________________________________________________________
separable_conv1d_1 (Separabl (None, 9, 256)            33664
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 4, 256)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 256)            0
_________________________________________________________________
global_average_pooling1d_1 ( (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 2052
=================================================================
Total params: 208,772
Trainable params: 208,772
Non-trainable params: 0
```

Figure 57: Depth Separable 1D convolutional model summary

2. 2D Convolutional Neural Networks

Convolutional neural networks are proven to be the most popular architectures for object detection and classification of images. Convolution layers are designed for finding patterns a.k.a features within the images by sliding the filter window over the input images. Recently CNN's have been tested with other data formats such as text and audio signals because of their innate capability of detecting patterns.

A simple sequential model architecture with 3 convolutional layers stacked with max-pooling layers, dropout layers, and dense output layers is used for audio classification. The MFCC features are also two-dimensional which can be assumed as the height and width of the image. No channels are considered as 1 as all the audio signals are mono channeled.

To utilize the benefits of batch training, all the audio signals are transformed into a uniform shape. To do so, the length of the audio is fixed as 5s as more than 95% of the respiratory data is below 5s. The audio files with duration more than 5s are truncated and files that are shorter than 5s are padded with zeros. So, the shape of the input to the conv2D layer is (20, 79, 1) where 20 is the number of MFCC features and 79 is the number of frames taking padding into account. The number of filters in each layer is increased from 32, 64 to 128. The kernel window size is set to 3. Relu activation function is used for all the convolutional layers and dense layers. L2 regularization is used before the output layer to minimize overfitting. The below image shows the model summary of 2D convolutional architecture.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 18, 77, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 75, 64)        18496
_________________________________________________________________
activation (Activation)      (None, 16, 75, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 37, 64)         0
_________________________________________________________________
dropout (Dropout)            (None, 8, 37, 64)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 6, 35, 128)        73856
_________________________________________________________________
activation_1 (Activation)    (None, 6, 35, 128)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 17, 128)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 3, 17, 128)        0
_________________________________________________________________
flatten (Flatten)            (None, 6528)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1671424
_________________________________________________________________
dense_1 (Dense)              (None, 32)                8224
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 132
=================================================================
Total params: 1,772,452
Trainable params: 1,772,452
Non-trainable params: 0
```

Figure 58: 2D CNN model summary

3. Long Short Term Memory networks

Long short term memory networks are a particular type of recurrent neural networks [16] which can learn long term dependencies. The key difference between the recurrent network and convolutional network is that the recurrent networks are designed such that it can interpret the temporal components while making predictions. Unlike CNN's, the RNNs use the previous or upcoming data points to make current predictions by reusing the activations of previous or later nodes in the series to influence the outcome.

BiLSTM is capable of learning long term dependencies in both directions. This concept significantly increases the amount of information for the network to make predictions. The below image shows the BiLSTM architecture. The LSTM architecture requires input in the shape of (no.of samples, time steps, features). MFCC features extracted from the audio files using the librosa module are reshaped accordingly. The input layer is followed by a masking layer through which the padded values are masked which facilitates the model to train only on the real data without padded values.

A normalization technique is used to normalize the inputs. The key difference between layer normalization and batch normalization is that in layer normalization inputs are normalized across features instead of the batch dimension. In layer normalization, the mean and standard deviation are computed independently of other samples in the mini-batch. The test results show that the layer normalization performs slightly better in recurrent networks. But since the tflite conversion results in an error [17] , batch normalization is applied. The concept of skip connection to avoid vanishing gradient problem in deep networks is utilized by concatenating the low-level features using the dense layer to the output of the previous layer. These layers are followed by pooling, flatten, and dense layers.

```
Model: "long_short_term_memory"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input (InputLayer)           [(None, 79, 20)]          0
_________________________________________________________________________________________________
masking_4 (Masking)          (None, 79, 20)            0           input[0][0]
_________________________________________________________________________________________________
Batchr_norm (BatchNormalization (None, 79, 20)         80          masking_4[0][0]
_________________________________________________________________________________________________
td_dense_tanh (TimeDistributed) (None, 79, 64)         1344        Batchr_norm[0][0]
_________________________________________________________________________________________________
bidirectional_lstm (Bidirection (None, 79, 256)        197632      td_dense_tanh[0][0]
_________________________________________________________________________________________________
skip_connection (Concatenate)  (None, 79, 320)         0           td_dense_tanh[0][0]
                                                                   bidirectional_lstm[0][0]
_________________________________________________________________________________________________
dense_1_relu (Dense)         (None, 79, 64)            20544       skip_connection[0][0]
_________________________________________________________________________________________________
max_pool_1d (MaxPooling1D)   (None, 39, 64)            0           dense_1_relu[0][0]
_________________________________________________________________________________________________
dense_2_relu (Dense)         (None, 39, 32)            2080        max_pool_1d[0][0]
_________________________________________________________________________________________________
flatten (Flatten)            (None, 1248)              0           dense_2_relu[0][0]
_________________________________________________________________________________________________
dropout (Dropout)            (None, 1248)              0           flatten[0][0]
_________________________________________________________________________________________________
dense_3_relu (Dense)         (None, 32)                39968       dropout[0][0]
_________________________________________________________________________________________________
softmax (Dense)              (None, 4)                 132         dense_3_relu[0][0]
=================================================================================================
Total params: 261,780
Trainable params: 261,740
Non-trainable params: 40
```

Figure 59: BiLSTM model summary

5.3.2 Fall Detection

Data Collection

MobiAct dataset is used for fall detection [13]. In this dataset, twelve activities of daily living are used along with 4 types of falls. The daily living activities noted are standing, walking, jogging, jumping, stairs up and down, standing to sit on chair transition, sitting on a chair, sitting on a chair to standing transition, stepping in and out of the car, and lying down. The falls are falling forward with hands hitting the ground, falling forward with knees hitting the ground, falling backward, and falling sideways.

#	Scenario Code	Scenario
1	BSC	Back-sitting-chair: Falling backward while trying to sit on a chair
2	CHU	Chair-Up: Transition from sitting on a chair to getting up and standing
3	CSI	Car-Step-In: Stepping to a car to sit
4	CSO	Car-Step-Out: Stepping out of a car
5	FKL	Front knees lying: Falling forward and knees hit the ground first
6	FOL	Forward Lying: Falling forward and hands hit the ground
7	JOG	Jogging
8	JUM	Jumping
9	SBE	The scenario of being exercise: Doing exercise

10	SBW	The scenario of being at work
11	SCH	Sit on chair: Transitioning from standing to sitting on a chair
12	SDL	Sideward lying: Falling sidewards
13	SIT	Sitting: Sitting on a chair
14	SLH	The scenario of leaving home
15	SLW	A scenario of leaving work
16	SRH	A scenario of returning home
17	STD	Standing
18	STN	Stairs down: Walking down the stairs
19	STU	Stairs up: Climbing the stairs up
20	WAL	Walking

Table 2: Fall Dataset Summary

Exploratory Data Analysis

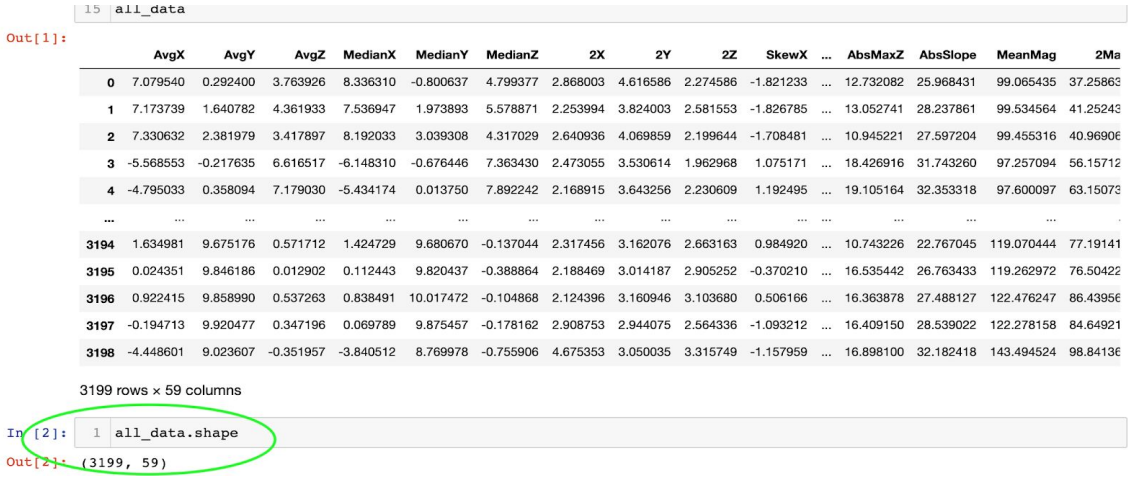
Data from the accelerometer and gyroscope sensors of a smartphone were recorded. Specifically, Lenovo K8 Note has been used to capture accelerometer and gyroscope sensor motion data. and several features have been calculated as shown in the figure[42]. Several scenarios are also mentioned in the dataset. A scenario is a collection of activities usually performed daily. For instance, the scenario of a person leaving home to office. This contains several activities mentioned earlier like standing, walking, stairs climbing, sitting in a chair, etc. Scenarios recorded in the dataset are a person leaving home, at work, leaving work, exercising, returning home. Data for these sensors for all 3 axes, x, y, and z are present.

```
avgX = sum(X)/len(X)
avgY = sum(Y)/len(Y)
avgZ = sum(Z)/len(Z)
medianX = median(X)
medianY = median(Y)
medianZ = median(Z)
stdX = stdev(X)
stdY = stdev(Y)
stdZ = stdev(Z)
skewX = skew(X)
skewY = skew(Y)
skewZ = skew(Z)
kurtosisX = kurtosis(X)
kurtosisY = kurtosis(Y)
kurtosisZ = kurtosis(Z)
minX = min(X)
minY = min(Y)
minZ = min(Z)
maxX = max(X)
maxY = max(Y)
maxZ  = max(Z)
slope = math.sqrt((maxX - minX)**2 + (maxY - minY)**2 + (maxZ - minZ)**2)
```

Figure 60: Triaxial features for fall data

Data Preprocessing

In this step, fall data is processed and transformed into csv files which can be used for training machine learning models. The data after preprocessing has dimensions as shown in the figure below,



```
In [15]: all_data
```

Out[1]:

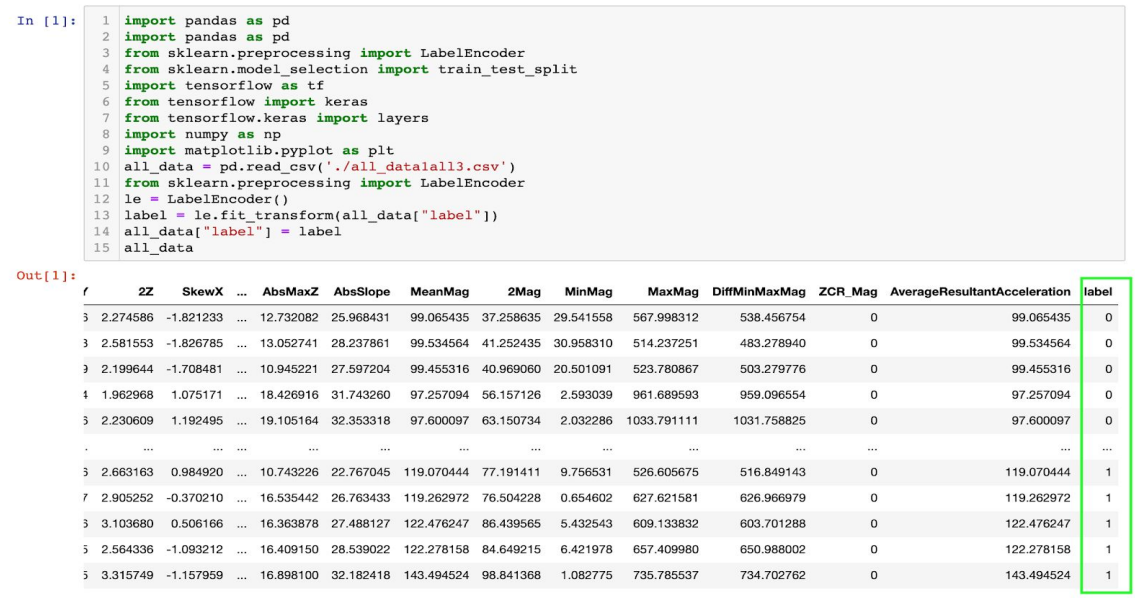
	AvgX	AvgY	AvgZ	MedianX	MedianY	MedianZ	2X	2Y	2Z	SkewX	...	AbsMaxZ	AbsSlope	MeanMag	2Ma
0	7.079540	0.292400	3.763926	8.336310	-0.800637	4.799377	2.868003	4.616586	2.274586	-1.821233	...	12.732082	25.968431	99.065435	37.25863
1	7.173739	1.640782	4.361933	7.536947	1.973893	5.578871	2.253994	3.824003	2.581553	-1.826785	...	13.052741	28.237861	99.534564	41.25243
2	7.330632	2.381979	3.417897	8.192033	3.039308	4.317029	2.640936	4.069859	2.199644	-1.708481	...	10.945221	27.597204	99.455316	40.96906
3	-5.568553	-0.217635	6.616517	-6.148310	-0.676446	7.363430	2.473055	3.530614	1.962968	1.075171	...	18.426916	31.743260	97.257094	56.15712
4	-4.795033	0.358094	7.179030	-5.434174	0.013750	7.892242	2.168915	3.643256	2.230609	1.192495	...	19.105164	32.353318	97.600097	63.15073
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3194	1.634981	9.675176	0.571712	1.424729	9.680670	-0.137044	2.317456	3.162076	2.663163	0.984920	...	10.743226	22.767045	119.070444	77.19141
3195	0.024351	9.846186	0.012902	0.112443	9.820437	-0.388864	2.188469	3.014187	2.905252	-0.370210	...	16.535442	26.763433	119.262972	76.50422
3196	0.922415	9.858990	0.537263	0.838491	10.017472	-0.104868	2.124396	3.160946	3.103680	0.506166	...	16.363878	27.488127	122.476247	86.43956
3197	-0.194713	9.920477	0.347196	0.069789	9.875457	-0.178162	2.908753	2.944075	2.564336	-1.093212	...	16.409150	28.539022	122.278158	84.64921
3198	-4.448601	9.023607	-0.351957	-3.840512	8.769978	-0.755906	4.675353	3.050035	3.315749	-1.157959	...	16.898100	32.182418	143.494524	98.84136

3199 rows × 59 columns

```
In [2]: all_data.shape
Out[2]: (3199, 59)
```

Figure 61: Dataframe dimensions for fall data

In the next step, the fall and no-fall labels are encoded to represent labels in a one-dimensional format.



```
In [1]:
import pandas as pd
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
all_data = pd.read_csv('./all_data1all3.csv')
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
label = le.fit_transform(all_data["label"])
all_data["label"] = label
all_data
```

Out[1]:

	2Z	SkewX	...	AbsMaxZ	AbsSlope	MeanMag	2Mag	MinMag	MaxMag	DiffMinMaxMag	ZCR_Mag	AverageResultantAcceleration	label
3	2.274586	-1.821233	...	12.732082	25.968431	99.065435	37.258635	29.541558	567.998312	538.456754	0	99.065435	0
3	2.581553	-1.826785	...	13.052741	28.237861	99.534564	41.252435	30.958310	514.237251	483.278940	0	99.534564	0
3	2.199644	-1.708481	...	10.945221	27.597204	99.455316	40.969060	20.501091	523.780867	503.279776	0	99.455316	0
4	1.962968	1.075171	...	18.426916	31.743260	97.257094	56.157126	2.593039	961.689593	959.096554	0	97.257094	0
3	2.230609	1.192495	...	19.105164	32.353318	97.600097	63.150734	2.032286	1033.791111	1031.758825	0	97.600097	0
.	...	...	...	...	...	...	...	...	...	...	...	...	...
3	2.663163	0.984920	...	10.743226	22.767045	119.070444	77.191411	9.756531	526.605675	516.849143	0	119.070444	1
7	2.905252	-0.370210	...	16.535442	26.763433	119.262972	76.504228	0.654602	627.621581	626.966979	0	119.262972	1
3	3.103680	0.506166	...	16.363878	27.488127	122.476247	86.439565	5.432543	609.133832	603.701288	0	122.476247	1
5	2.564336	-1.093212	...	16.409150	28.539022	122.278158	84.649215	6.421978	657.409980	650.988002	0	122.278158	1
5	3.315749	-1.157959	...	16.898100	32.182418	143.494524	98.841368	1.082775	735.785537	734.702762	0	143.494524	1

Figure 62: Encoded class labels 0 (no-fall) and 1 (fall)

Machine learning models

For fall detection analysis, different models were employed and 80% training and 20% testing holdout set criteria were used.

1. Support Vector Machine

SVM is supervised machine learning models that are used to analyze data and classify labeled data. In this project, SVM was used to train the fall data and the dataset is split into training and testing sets.

```
In [18]:
# Training dataset using Support vector machine
from sklearn.model_selection import train_test_split
from sklearn import svm
X_train, X_test, y_train, y_test = train_test_split(_all_data_x, _all_data_y, test_size=0.2, random_state=42)


clf = svm.SVC(gamma='scale')
clf.fit(X_train, y_train)


test_count = len(X_test)

print(test_count)
```

```
640
```

Figure 63: Training dataset using SVM

2. K-Nearest Neighbor Model

KNN is the simplest yet proven algorithm for the classification of a new data point. This algorithm is based on feature similarity. In this project, KNN was used to train the fall data and exceptional accuracy has been achieved from this algorithm. The figure below shows the training of the fall dataset.

```
In [3]:
# KNN model to train fall dataset
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
X_train, X_test, y_train, y_test = train_test_split(_all_data_x, _all_data_y, test_size=0.2, random_state=42)


clf = KNeighborsClassifier()
clf.fit(X_train, y_train)


test_count = len(X_test)

print(test_count)
```

```
640
```

Figure 64: Training dataset using KNN

3. Keras Sequential Model

Keras' sequential model was used to train the fall dataset. First of all, a basic sequential model was declared. Next, the input layer was declared with the input shape parameter. Later, 2 convolution layers have been added to the model. For dense layers, the first parameter was the output size of the layer. At the output layer, sigmoid activation function is used to get probability of fall or no-fall. To compile the model loss function binary- cross entropy and adam optimizer was declared. The below figure shows the declaration of the Keras sequential model.

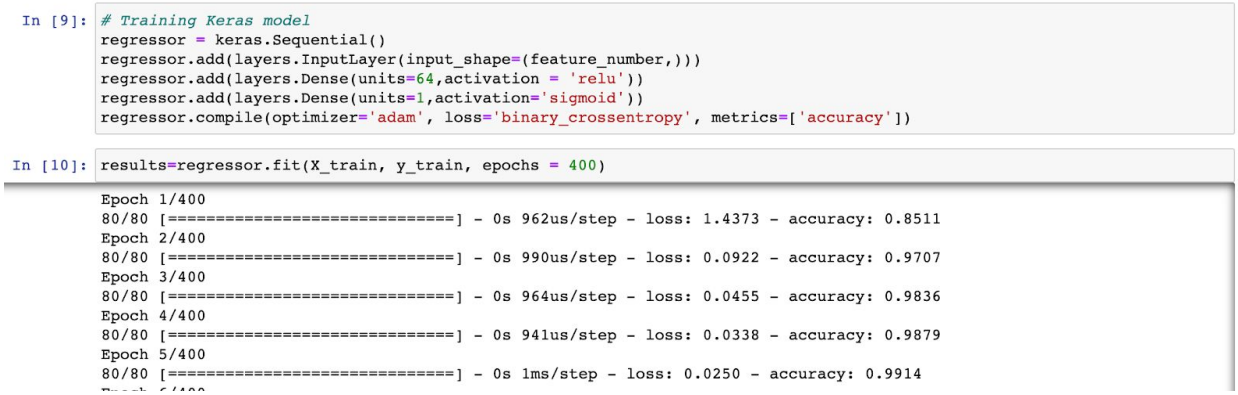


Figure 65: Declaration of Keras Sequential model

5.3.3 Tflite Conversion

After comparing the performances of the selected models for respiratory check and fall detection, the best performing one was chosen for deployment. The trained weights of the model with higher classification accuracy was loaded using keras.models.load_model() method. and converted to a tflite file using tf.lite.TFLiteConverter.from_keras_model() method. The converted model is then saved as a .tflite file which can be loaded and interpreted using tf.lite.Interpreter for making predictions.

Chapter 6. Testing and Verification

6.1 Functional and UI Testing

In this project, JUnit is used for instrumentation testing in which android and web application-specific functionalities such as activities, fragments, and services will be tested. Espresso testing framework is used for android application UI testing as it allows us to simulate every possible action a user might perform with the application. Also, it allows us to test the application at different speeds than a normal user ever would. These tests ensure that the UI doesn't break the application. Both instrumentation and UI tests will be performed on an emulator or a real device.

UI testing has been done to satisfy the following test cases,

#	Test Case	Expected output
1.	The patient should be able to login successfully with a valid password and username	Successful login
2.	The patient should be able to upload respiratory sound (crackle and wheezes)	Respiratory data uploaded successfully
3.	The patient should be able to view the profile dashboard with all the details	Records and present reports should be visible on the dashboard
4.	The patient should be able to classify his/her respiratory sound	The classification model should classify crackles or wheezes or normal in the given period

5.	The patient should be able to log out successfully	Successful logout
6.	The system should collect data using IoT devices in the application	Successfully able to collect data
7.	The system should have ability to work with no network connectivity	Able to connect in remote areas
8.	The recommendation system should send notification to primary contact	Recommendation system is able to notify primary contact
9.	Mobile application should be able to detect no-fall when a person is sitting	No-fall is detected when a person is simply sitting
10.	Mobile application should be able to detect fall when it is fall	Fall is detected and notification is sent to the primary doctor

Table 3: Test cases and expected output

6.2 End-to-end Testing

In the end-to-end testing, the complete application flow starting from the data collection part to prediction and recommendation will be tested. For the respiratory illness detection use case, due to the unavailability of the stethomicrophone [11], the test samples from the dataset are transferred to the emulator storage and the predictions are verified.

For the fall detection use case, the sensor data collected from different mobile devices models will be cross verified with the training data. The application will be tested by keeping the mobile phone in different positions such as trouser pocket, shirt pocket, sitting, moving, etc to make sure that the model can predict the occurrence of falls. The application has been tested over one

hundred test scenarios in different motion situations like, 1) sitting posture, 2) ascending or descending stairs, 3) talking while walking, 4) jogging, and 5) jumping.

6.3 Non-functional Testing

1. Usability: Usability testing is to be done and the application is available through the web and user interfaces.
2. Reliability and capacity for the web application: Even when there are more than a thousand requests simultaneously, the application should work and provide recommendations to the users.
3. Reliability and capacity for the mobile application: In the mobile application, only one request can be processed at a time. A serial set of at least a hundred requests should work and provide recommendations to the user.
4. Error handling: In the event of an error in the application, it should be handled and the user notified of an issue. A component's failure should not crash the application.
5. Performance of web application: The web application should provide recommendation results to the user in under 5 seconds on an average.
6. Performance of the mobile application: The mobile application should provide recommendation results to the user in under 10 seconds on average.
7. Supportability of web application: The web application should be compatible with commonly used browsers. It should work with at least Chrome version 40 and higher, Internet Explorer version 11 and higher, Mozilla Firefox version 40 and higher, and Safari version 5 and higher.
8. Supportability of mobile application: The mobile application should be compatible with Android versions 6 and higher.
9. Security: In the web application that is used by multiple users, for security, access control is implemented. Authentication and authorization rules should be enforced. Users require a password to access their data.
10. Scalability: The web application should be easily scalable to accommodate a large number of users. The machine learning model should be able to process large volumes of data.

11. Latency: The application should provide results and send notifications with a latency of fewer than 5 secs.

Chapter 7. Performance and Benchmarks

7.1 Performance of Machine Learning Models

This project requires two machine learning models. One is for crackles/wheezes detection and the other is for fall detection and classification. The performance results of different models compared are listed below.

Respiratory illness detection

The following metrics for evaluating the classification results were defined by the authors of the ICBHI challenge dataset.

- Sensitivity (SE) = $(Cc + Ww + Bb)/(C + W + B)$
- Specificity (SP) = Nn/N
- Average score(AS) = $(SE + SP)/2$
- Harmonic score (HS) = $(2 * SE * SP)/(SE + SP)$

Where,

- N - Total number of normal sounds
- Nn - Number of samples rightly classified as normal
- C -Total number of crackle sounds
- Cc - Number of samples rightly classified as crackles
- W - Total number of wheeze sounds
- Ww - Number of samples rightly classified as wheezes
- B - Total number of sounds that contain both crackle and wheezes
- Bb - Number of samples rightly classified as both crackles and wheezes

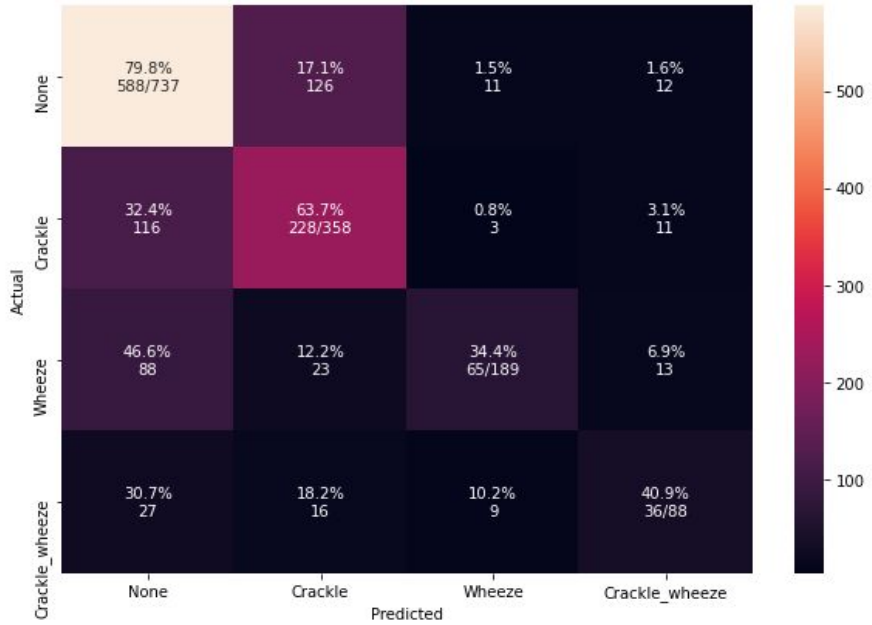
Specificity is defined as the model's ability to identify normal healthy sounds. The sensitivity of the model is defined as the model's ability to predict abnormal sounds.

The below table shows the comparison results of 1D separable convolution, 2D convolution, and LSTM networks.

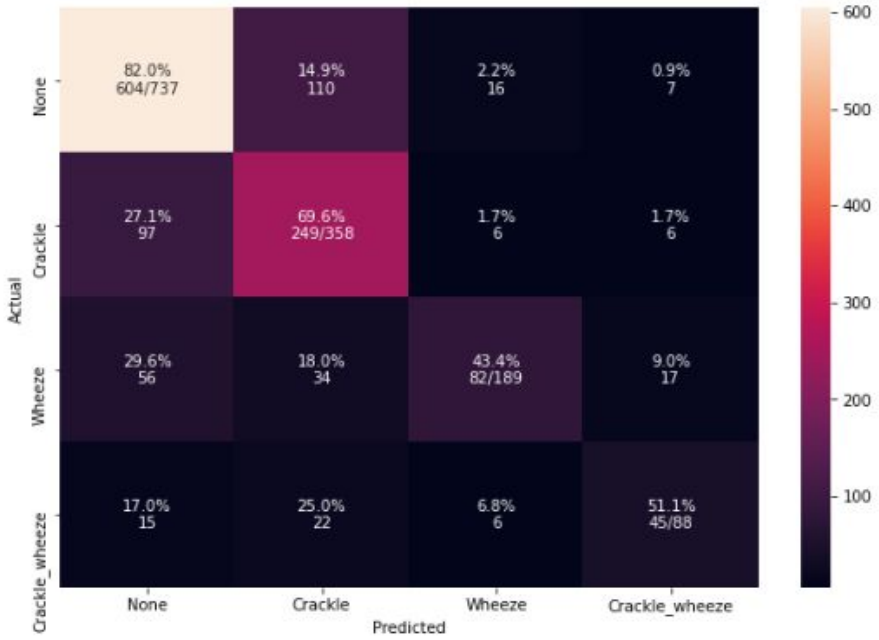
Model	**Specificity**	**Sensitivity**	**Average Score**	**Harmonic Score**
1D Separable Conv	0.822	0.459	0.640	0.589
2D CNN	0.798	0.518	0.658	0.628
BiLSTM	**0.82**	**0.60**	**0.71**	**0.70**

The confusion matrix of each model is shown below.

1)



2)



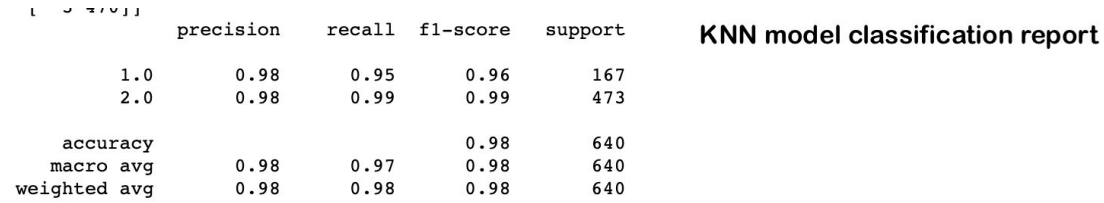
3)

Figure 66: Confusion matrix 1) 1D convolution, 2) 2D CNN, 3) LSTM models

Overall LSTM model performed better in terms of sensitivity and specificity; hence it was chosen for deployment.

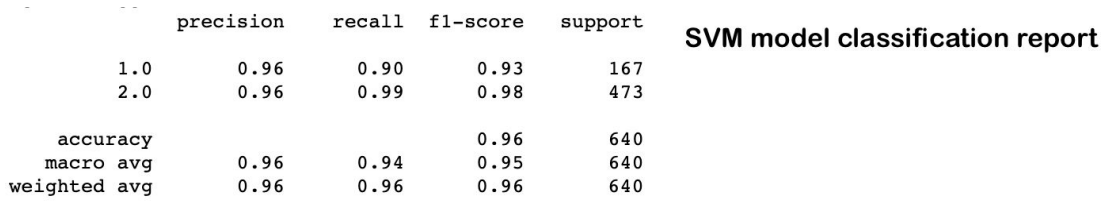
Fall detection

On the other hand, fall data is trained using three machine learning algorithms K- nearest neighbors (KNN, figure 67), support vector machine (SVM, figure 68), and Keras sequential model(figure 69). The classification report of the models is shown in the below snapshots.



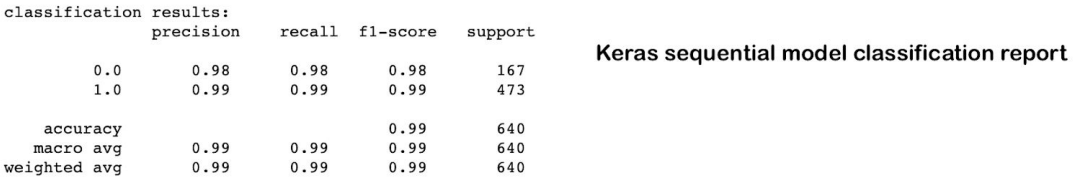
	precision	recall	f1-score	support
1.0	0.98	0.95	0.96	167
2.0	0.98	0.99	0.99	473
accuracy			0.98	640
macro avg	0.98	0.97	0.98	640
weighted avg	0.98	0.98	0.98	640

Figure 67: KNN model classification report



	precision	recall	f1-score	support
1.0	0.96	0.90	0.93	167
2.0	0.96	0.99	0.98	473
accuracy			0.96	640
macro avg	0.96	0.94	0.95	640
weighted avg	0.96	0.96	0.96	640

Figure 68: SVM model classification report



classification results:

	precision	recall	f1-score	support
0.0	0.98	0.98	0.98	167
1.0	0.99	0.99	0.99	473
accuracy			0.99	640
macro avg	0.99	0.99	0.99	640
weighted avg	0.99	0.99	0.99	640

Figure 69: Keras sequential model classification report

This project used Keras sequential model for fall detection as it provided 99% accuracy.

Chapter 8. Deployment, Operations, Maintenance

The web application built using ReactJs will be deployed in the cloud, this would be done using the Heroku platform. Heroku is a platform as a service that enables applications to deploy, run, and maintain in the cloud. It is also one of the first cloud platforms and has been there since 2007.

To automate the deployment process CICD has been used. CICD stands for Continuous Integration and Continuous Deployment. To achieve this we use Github actions and Heroku. Github actions enable the automation of software development using Git.

This pipeline would enable us to directly deploy the web application as soon as the latest code is pushed to the remote Github repository. The pipeline will run test cases against the latest code that has been pushed before deploying and will only finish deploying if all the test cases have passed. If the latest commit does not pass all the test cases the deployment will be reverted and the last successful commit will be deployed.

Chapter 9. Summary, Conclusions, and Recommendations

9.1 Summary and Conclusions

This project proposes and builds an AI-enable IoT edge for rural and high-risk patients particularly narrating the issues with respiratory illness and elderly falls. The project consists of multiple modules to bring together this solution. A Machine Learning pipeline to train and build models that can be deployed on an IoT edge device like a smartphone. An Android app that runs machine learning models locally to predict the users' health. A web application to let the users visualize and analyze their health statistics.

To perform a respiratory check the user will have to select the audio file to predict. Upon selecting the audio file the machine learning model will predict the users' respiratory condition. The result of the check is also synchronized with a web application with the help of a real-time database Firebase Cloud Firestore.

Similarly, when the user needs to perform fall detection they have to launch the Fall detection activity so that the application reads sensor data and feeds it to the machine learning model to make a fall prediction. Upon prediction, the data is synchronized to a web application with the help of a real-time database Firebase Cloud Firestore.

Finally, The web dashboard enables a user to analyze and visualize their health statistics.

9.2 Recommendations for Further Research

Few recommendations for future reach are as follows:

- Collecting and training diverse respiratory sound data including children and elders for building a robust model
- Taking in live user breathing audio to analyze respiratory health of the user
- Extending fall detection from smartphone to IoT wearable device

- Adding additional features in the web dashboard that will enable a medical professional to reach out to the user in case of emergency
- For the fall model, various devices can be used to collect sensor data and training purposes.

Glossary

Adaptive AI	Adaptive artificial intelligence process monitors and learns the new changes made to the input and output values and their associated characteristics.
Crackle	Crackles are abnormal lung sounds characterized by discontinuous clicking or rattling sounds. Crackles can sound like salt dropped onto a hot pan or like cellophane being crumpled or like velcro being torn open
Wheeze	A wheeze is a continuous, coarse, whistling sound produced in the respiratory airways during breathing.
Heuristic AI	Heuristic AI is based on cognitive science or the study of the human mind.
k-Nearest neighbor (k-NN)	k-nearest neighbor is a type of instance learning algorithm and it is used for classification and regression data analysis.
ML	Machine learning is an application of AI that provides systems the ability to automatically learn.
Neural Networks (NN)	Neural networks are a set of algorithms, modeled loosely based on the human brain and used to recognize patterns.

Support Vector Machine (SVM)	Support vector machines are supervised machine learning models. It can analyze data used for both classification and regression analysis.
LSTM	Long short-term memory is an artificial recurrent neural network architecture which has feedback connections which makes them process sequence data
CNN	Convolutional neural network is a class of deep learning architecture where convolution operation is performed in place of general matrix multiplication. Most commonly used in image data classification, object detection and segmentation tasks.

References

[1] CDC.gov, "Urban-Rural Differences in COPD", *cdc.gov*, 2020. Available [Online]: https://www.cdc.gov/copd/features/copd-urban-rural-differences.html

[2] McClure, Hughes, Ren, et al, "The population approach to falls injury prevention in older people: findings of a two community trial", *BMC Public Health*, Vol.10, no.79, 2010. Available [Online]: https://bmcpublichealth.biomedcentral.com/articles/10.1186/1471-2458-10-79

[3] CDC.gov, "Important Facts about Falls", *cdc.gov*, 2017. Available [Online]: https://www.cdc.gov/homeandrecreationalsafety/falls/adultfalls.ht

[4] T. Davenport, and R. Kalakota, "The Potential for Artificial Intelligence in Healthcare". *Future Healthcare Journal*, Vol. 6, no. 2, pp. 94-98, June 2019. Available[Online]: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6616181/pdf/futurehealth-6-2-94.pdf

[5] J. Guo, and B. Li, "The Application of Medical Artificial Intelligence Technology in Rural Areas of Developing Countries", *Health Equity*, Vol. 2, no. 1, pp. 174-181, August 2018. Available[Online]: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6110188/pdf/heq.2018.0037.pdf

[6] H. Mora et al. "An IoT-Based Computational Framework for Healthcare Monitoring in Mobile Environments", *Sensors(Switzerland)*, Vol. 17, no. 10, pp. 2302, October 2017. Available[Online]: https://doaj.org/article/77c05c8e9a75446fb89e72f00821741e

[7] S. Yoo, and D. Oh, "An Artificial Neural Network-based Fall Detection." *International Journal of Engineering Business Management* Vol. 10 pp. 184797901878790, 2018. Available[Online]: https://doi.org/10.1177/1847979018787905

[8] A. Ramalingam et al. "IEEE FEMH Voice Data Challenge 2018", IEEE International Conference on Big Data(Big Data), pp. 5271-5276, December 2018. Available[Online]: https://ieeexplore-ieee-org.libaccess.sjlibrary.org/document/8622164

[9] M Gronnesby et al. “Features Extraction for Machine Learning-Based Crackle Detection in Ling Sounds from a Health Survey”, 2017. Available[Online]: https://arxiv.org/abs/1706.00005

[10] C. Vuppalapati, “Democratization of Artificial Intelligence(AI) for the future of Humanity”, *CRC book publisher,* Vol.1.

[11] https://www.etsy.com/listing/477257401/stethomicrophone

[12] Rocha BM et al. (2019) "An open-access database for the evaluation of respiratory sound classification algorithms" Physiological Measurement 40 035001

[13] bmi.hmu.gr, “The MobiFall and MobiAct datasets”, *bmi.hmu.gr*. Available [Online]: https://bmi.hmu.gr/the-mobifall-and-mobiact-datasets-2/

[14] T. Giannakopoulos and G. Pavan, "pyAudioAnalysis: An Open-Source Python Library for Audio Signal Analysis," Plos One, vol. 10, (12), pp. e0144610, 2015. . DOI: 10.1371/journal.pone.0144610.

[15] S. Gupta et al, "Feature extraction using MFCC," Signal & Image Processing: An International Journal (SIPIJ), vol. 4, (4), pp. 101-108, 2013.

[16] D. Perna and A. Tagarelli, "Deep auscultation: Predicting respiratory anomalies and diseases via recurrent neural networks," in 2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS), 2019.

[17] https://github.com/tensorflow/tensorflow/issues/43934#issuecomment-714832778

[18] A Mondal et al. “Detection of Lungs Status Using Morphological Complexities of Respiratory Sounds”, *TheScientificWorld,* pp. 182938-9, February 2014. Available[Online]: https://doaj.org/article/84aad15ce62c4e75bd7308ba7742fc56

[19] T Grzywalski et al. “Practical Implementation of Artificial Intelligence Algorithms in Pulmonary Auscultation Examination”, *European Journal of Pediatric*, Vol. 178, no. 6, pp. 883-890, June 2019. Available[Online]: https://link-springer-com.libaccess.sjlibrary.org/article/10.1007%2Fs00431-019-03363-2

[20] Z Ren et al, "Identifying Tuberculous Pleural Effusion Using Artificial Intelligence Machine Learning Algorithms", *Respiratory Research*, Vol. 20, no. 1, pp. 220-229, October 2019. Available[Online]: https://doaj.org/article/09cf889a46374f7e93dab3f8585e99a

[21] P Porter et al, "A Prospective Multicentre Study Testing the Diagnostic Accuracy of an Automated Cough Sound Centred Analytic System for the Identification of Common Respiratory Disorders in Children", *Respiratory Research*, Vol. 20, no. 1, pp. 81-110, June 2019. Available[Online]: https://doaj.org/article/76a4eef7cc61453695ea0de6aa873aeb

[22]Dhiraj et al. "An Effective Analysis of Deep Learning-Based Approaches for Audio Based Feature Extraction and Its Visualization", *Multimedia Tools and Applications*, Vol. 78, no.17, pp. 23949-3972, September 2019. Available[Online]: https://link-springer-com.libaccess.sjlibrary.org/article/10.1007/s11042-018-6706-x

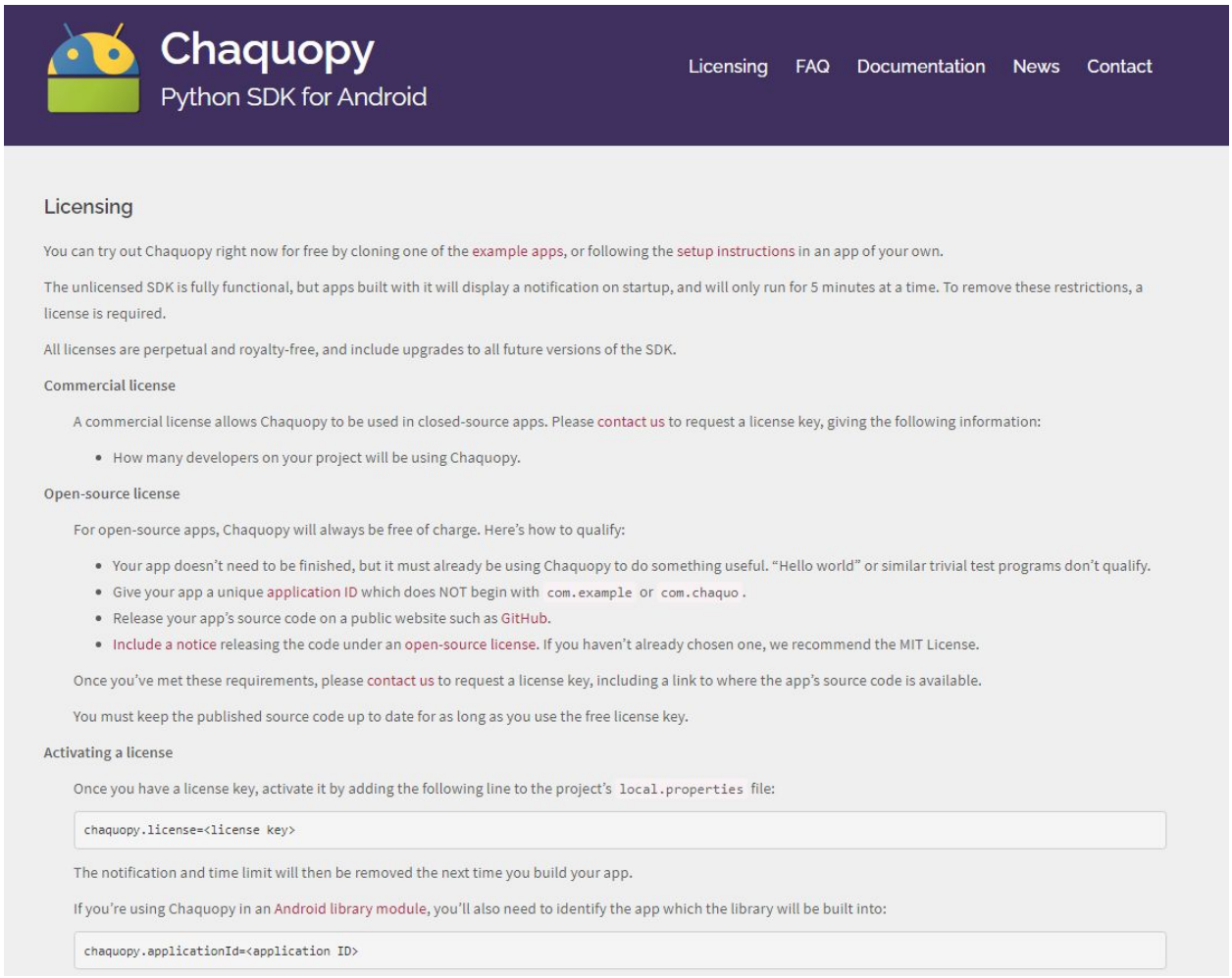
Appendices

Appendix A: Source Code

Github repository:
https://github.com/priyanshi9692/Adaptive-and-Heuristic-AI-enabled-IoT-Edge-for-high-risk-and-rural-patients

Appendix B: Chaquopy License

Chaquopy enables the use of python code in Android studio. Over 90% of the top python packages can be installed using pip. The Chaquopy license can be requested for free by following the below steps mentioned at https://chaquo.com/chaquopy/license/.



Chaquopy
Python SDK for Android

Licensing FAQ Documentation News Contact

Licensing

You can try out Chaquopy right now for free by cloning one of the example apps, or following the setup instructions in an app of your own.

The unlicensed SDK is fully functional, but apps built with it will display a notification on startup, and will only run for 5 minutes at a time. To remove these restrictions, a license is required.

All licenses are perpetual and royalty-free, and include upgrades to all future versions of the SDK.

Commercial license

A commercial license allows Chaquopy to be used in closed-source apps. Please contact us to request a license key, giving the following information:

- How many developers on your project will be using Chaquopy.

Open-source license

For open-source apps, Chaquopy will always be free of charge. Here's how to qualify:

- Your app doesn't need to be finished, but it must already be using Chaquopy to do something useful. "Hello world" or similar trivial test programs don't qualify.
- Give your app a unique application ID which does NOT begin with `com.example` or `com.chaquo`.
- Release your app's source code on a public website such as GitHub.
- Include a notice releasing the code under an open-source license. If you haven't already chosen one, we recommend the MIT License.

Once you've met these requirements, please contact us to request a license key, including a link to where the app's source code is available.

You must keep the published source code up to date for as long as you use the free license key.

Activating a license

Once you have a license key, activate it by adding the following line to the project's `local.properties` file:

```
chaquopy.license=<license key>
```

The notification and time limit will then be removed the next time you build your app.

If you're using Chaquopy in an Android library module, you'll also need to identify the app which the library will be built into:

```
chaquopy.applicationId=<application ID>
```